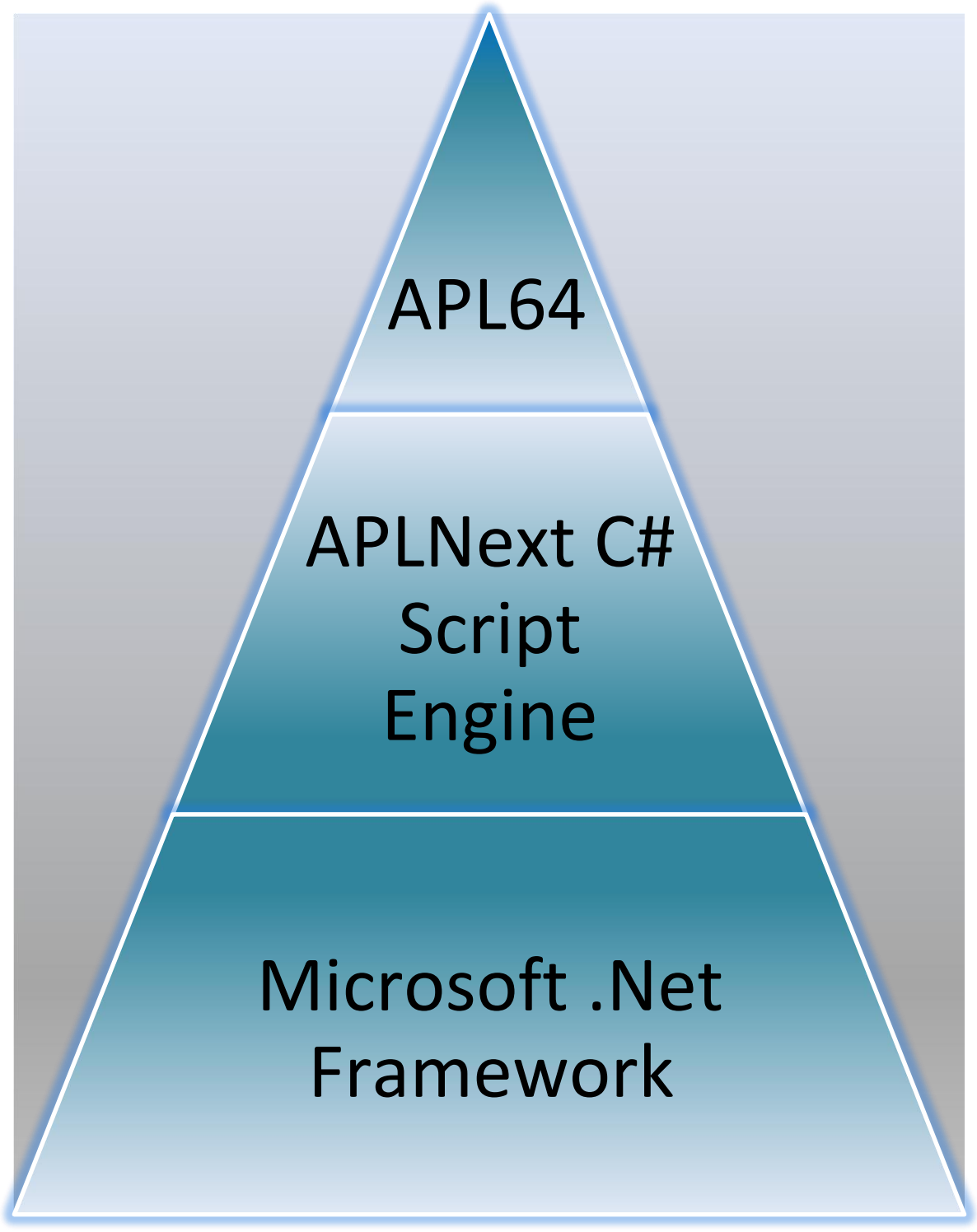

APL64
APLNext C# Script Engine
Microsoft .Net Framework

Contents

Overview – Easily Use C# and .Net from an APL64 Application

Microsoft .Net is a colossal, well-documented and easy-to-use application programming interface (API) toolkit which can be used in the Windows, Android, Linux and iOS operating systems in desktop, server, browsers and other environments. The Microsoft C# programming language is the premier tool used world-wide by millions of programmers to access and use the Microsoft .Net. Ready-to-run, sample C# source code is available without cost from Microsoft and other sources on the Internet for virtually all application system development needs.

APL64 and the APL64 C# Script Engine are based on Microsoft .Net and Microsoft .Net Standard. The APL64 C# Script Engine does not provide an interface to the Microsoft-deprecated .Net Framework.

The C# statements necessary to accomplish a task in .Net are easy-to-understand, 'English language', object-oriented statements in contrast to the numerous special 'codes', 'structures' and 'buffers' that may have been necessary in the past.

The □cse system function is implemented as the interface to the APLNext C# Script Engine (CSE). With the CSE, an APL64 application system programmer can use the C# programming language to access the Microsoft .Net and obtain the benefit of the Framework tools. This document describes the CSE object model, including its methods, properties and events as well as examples illustrating the use of the CSE from APL64.

A few of the projects which can be implemented using the CSE:

- Explore .Net objects and run C# statements line-by-line
- Create, save, edit and run CSE scripts to use the full .Net
- Create utilities and modules using .Net features for use in APL64 application systems
- Create C# classes, properties and methods for use in APL64 applications
- Use .Net web tools to access and capture web-based information
- Use Ado.Net to access databases such as Microsoft Access and SQL Server, Oracle and IBM DB2
- Use Microsoft Linq (to SQL, to Objects, to XML) to handle data using queries
- Use .Net tools to create, parse, edit and update data in Csv, Xml, Json and OData formats
- Access Microsoft Office using .Net tools
- Use 3rd-party .Net tools like SpreadsheetGear, GemBox and DynamicPdf in their native C# format using the APL64 □cse system function.

Projects which cannot be implemented using the CSE:
Microsoft has deprecated the use of the Rosyln .Net compiler for projects which involve the Windows Presentation Foundation (WPF) or System.Windows.Forms (Winforms) or WPF GUI tools. Because the CSE is based on the Rosyln .Net compiler, events exposed by WPF or Winforms cannot be handled by the CSE. Using the CSE for WPF or Winforms GUI projects is not feasible.

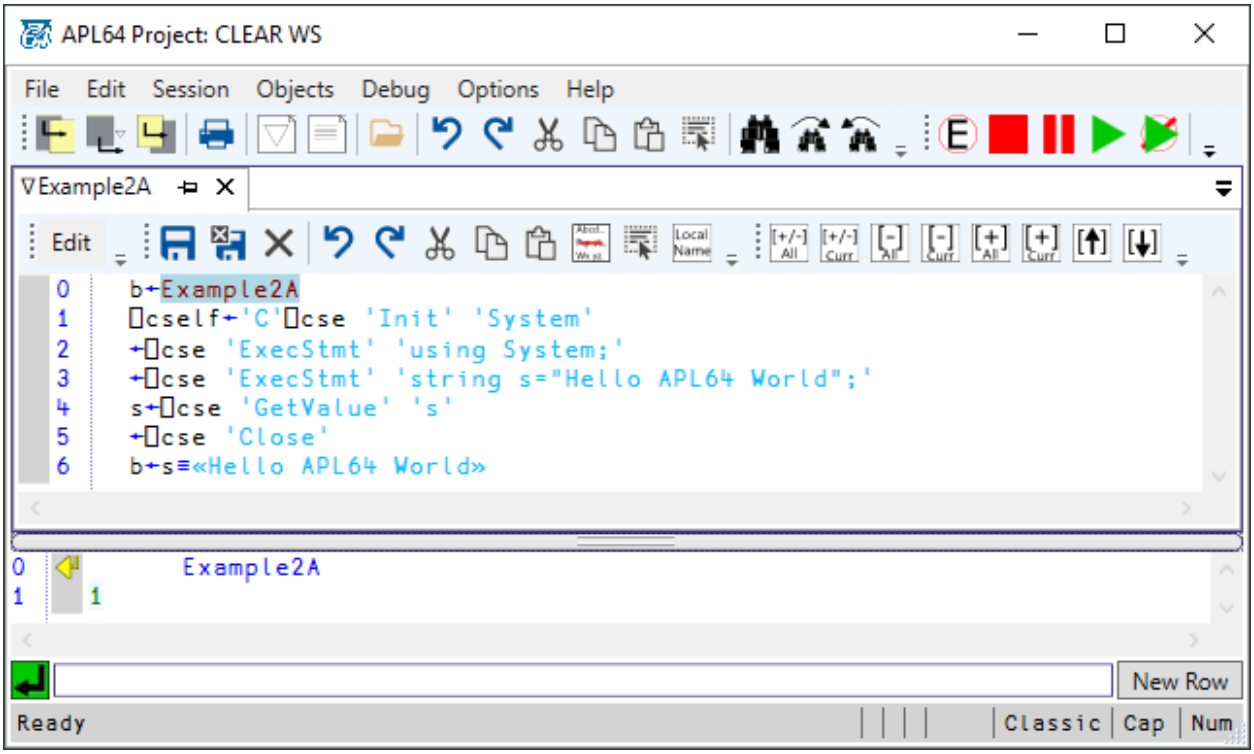
Hello World – Your First C# Program using the APL64 and the CSE
The following APL statements:

- Use the CSE 'Init' method to initialize an instance of the CSE object
- Use the CSE 'ExecStmt' method to:
- Define the 'System' prefix so that the subsequent C# statements are less verbose
- Modify the state of that CSE object to contain a C# string object 's'
- Use the CSE 'GetValue' method to get the value of the C# variable 's' into the APL64 workspace
- Use the CSE 'Close' method to close the CSE object instance.

Example #2A:

In this example the CSE ExecStmt method is used to set the value of the C# string variable s.

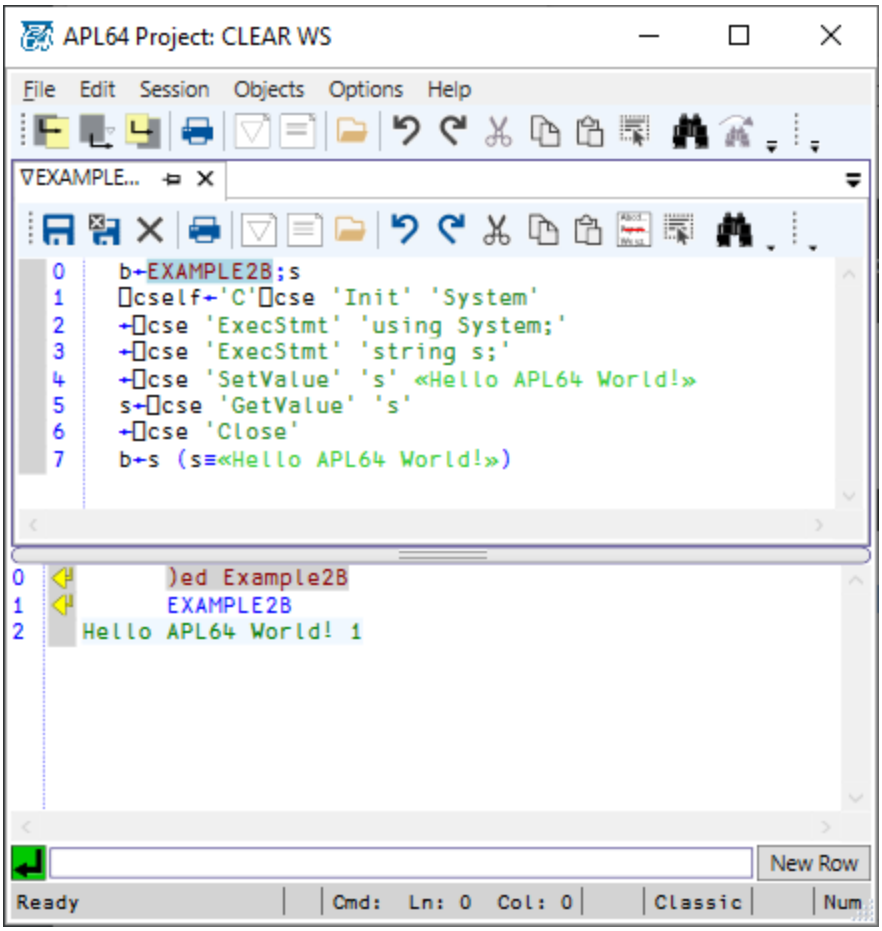
```
b←Example2A
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
←□cse 'ExecStmt' 'string s="Hello APL64 World";'
s←□cse 'GetValue' 's'
←□cse 'Close'
b←s≡«Hello APL64 World»
```



Example #2B:

In this example the CSE 'SetValue' method is used to set the value of the C# string variable s from an APL64 string value.

```
b←EXAMPLE2B;s
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
←□cse 'ExecStmt' 'string s;'
←□cse 'SetValue' 's' «Hello APL64 World!»
s←□cse 'GetValue' 's'
←□cse 'Close'
b←s (s≡«Hello APL64 World!»)
```



APL64 □cse System Function

The APL64 □cse system function is used to access the CSE system object which applies to all CSE instances created during an APL64 session. The APL64 application programmer may create multiple CSE instances, each of which is contained with the same CSE object for the current session. The □cse system function accessing the CSE object has the following syntax:

result ← '#' □cse *actionName* [*arguments*]

Syntactic Element	Description
result	APL64 value which is result, if any, of the action performed
#	Indicates that the CSE action will be performed on the CSE object
actionName	Name of the CSE action to be performed on the CSE object
arguments	APL64 Value(s), if any, which are required by a particular CSE action

The APL64 □cse system function is also used to create and control CSE instances based on the object model of the CSE. The action of multiple uses of the □cse system function on a specific instance of the CSE object is cumulative. The □cse system function accessing a CSE instance has the following syntax:

result ← *instanceNAME* □cse *actionNAME* [*arguments*]

Syntactic Element	Description
result	APL64 value which is result, if any, of the action performed
instanceName	Name of the CSE instance on which the action will be performed. The instanceName is optional if the □cself system variable value has been set.
actionName	Name of the CSE action to be performed on the CSE instance
arguments	APL64 Value(s), if any, which are required by a particular CSE action

CSE Technology

The APL64 □cse system function provides a convenient C# portal to .Net. The APL64 □cse system function enables an APL programmer to initialize and manipulate the CSE object and its instances. In a CSE instance, the APL64 programmer can execute text scripts which contain one or more valid C# programming statements. CSE scripts may also load file-based .Net assemblies and use the object model of .Net classes in those assemblies. A CSE script is automatically debugged, compiled, and executed using the .Net C# debugger and compiler by the CSE technology.

The state of a CSE object is modified via communication between the CSE object and its instances and APL64 environment which initiated the CSE using the object model of the CSE. A CSE object will persist until:

- The APL64 environment which initiated the CSE is ended, or
- The APL64 instance uses the CSE 'Close' method, or
- The CSE instance is replaced by another CSE instance of the same name

When a CSE script is being executed by an instance of the CSE object, the APL64 environment which initiated this action has effectively transferred execution control to the CSE object until the execution of CSE script has completed. While a multi-line CSE script is being executed by an instance of the CSE, execution control can be temporarily switched back to the initiating APL64 environment from within that CSE script by incorporating an 'APL:…' script statement in the script.

C# scripts can create .Net objects, .Net methods operating on those objects and .Net events fired by those objects. Besides executing C# scripts, APL64 can also set or obtain the value of .Net object properties created in a CSE instance. APL64 can execute .Net methods using arguments provided by APL64 and receive results provided by those .Net methods.

Multiple instances of the CSE object can be created by APL64 and inter-CSE object communication is possible through the initiating APL64 environment.

The state of a CSE object includes instances of .Net objects such as value types like Int32, Double and String which correspond to familiar APL64 data types as well as more complex .Net objects such as DateTime instances and custom .Net class types.

- The CSE and APL64 are 64-bit processes

- The APL64, and hence the CSE, can access memory over 2GB and create multiple .Net objects each over 2GB, up to the limit of the memory on the target workstation, currently 192GB using the Windows 7 Professional operating system

When an instance of the CSE object is created in an APL64 instance, a CSE engine is created which can execute C# source code provided by the APL64 programmer.

Installation and Deployment of the CSE

CSE Requirements

The requires:

- APL64
- Use 2⊃□sysinit to determine the current .Net version for APL64 and the APL64 C# Script Engine.
- 64-bit workstation hardware

CSE Installer

The APL64 CSE is an integral part of the APL64 interpreter, so no separate APL64 CSE installer is needed.

APL64 xml-format File Configuration of the CSE

The 'CseObjectOptions' and 'CseInstanceOptions' sections of the APL64 xml-format configuration file contains default configuration information which applies to when the CSE object and any CSE instance is created. These sections of the APL64 configuration file are not updated when the value of one of these properties is modified under program control during the APL64 session or when APL64 configuration settings are saved.

Setting	Location	Configuration File Value	APL64 Programmer Information
ReturnOnError	CseInstanceOptions	Default false	CSE instance 'returnonerror' property value: 0/False 1/True
trackevents	CseInstanceOptions	Default false	CSE instance 'trackevents' property value: 0/False 1/True
UseReplStr	CseInstanceOptions	Default false	CSE instance 'usereplstr' property value: 0/False 1/True

CSE FAQ

What is the CSE?

The CSE is the APLNext C# Script Engine for APL64 exposed via the APL64 □cse system function. This is an easy-to-use interface between APL64 and the Microsoft .Net which uses the C# programming language. Using the object model of the CSE (methods, properties and events), an APL64 programmer can create instances of a .Net environment which is in a separate memory space from the APL64 memory space. In such a CSE instance .Net objects, such as familiar value types (Int32, double, string) as well as more complex .Net objects will be directly accessible to the APL64 environment.

The CSE empowers the APL64 programmer with the benefits of the Microsoft .Net in an enhanced version of APL64 which maintains full compatibility with prior versions of APL64.

.Net provides a myriad of powerful tools and features which are easy to use and fully documented. The APLNext C# Script Engine simplifies the use of .Net without the need to use Microsoft Visual Studio. The APL64 □cse system function provides a convenient syntax for using the APLNext C# script engine without imposing .Net principles on APL64.

What is included in the CSE?
The CSE software package is available to current APL64 subscribers and includes:

- The APL64 developer and runtime executables with the □cse system function
- The APLNext CSE .Net component installer
- The CSE pdf-format documentation
- The CSE 'CSE Code Samples' folder, delivered as a .zip file
- The APL2000 Forum section for the CSE
- Access to the APL2000 Support service

How does APL2000 distribute the CSE to APL64 product subscribers?
The CSE is included in every version of APL64. The developer and run-time versions of APL64 support the □cse system function.

When did the CSE become a production feature of the APL64 product?
The initial version of the APL64 product contained the CSE.

Can the CSE be deployed as part of an APL64 runtime application?
Yes. There are no anticipated additional APL2000 license requirements or cost to APL64 product subscribers to incorporate CSE technology in their APL64 runtime applications.

Can the CSE technology be used with any developer or runtime version of APL64?
Yes. The APL64 CSE in an in-process engine and part of the APL64 interpreter. Each version of APL64 incorporates an appropriate version of the APL64 CSE.

The CSE development team strives to maintain an invariant programming syntax among CSE versions, however new features of the CSE may require new programming syntax. Maintaining a current APL64 product subscription is the way to always have the current APL64 technology available to programmers and end users of an APL64-based application system.

What are the license provisions of the developer and runtime APL64 components?
When the APL64 developer component is installed, the "APL64 End User License Agreement" (EULA) is also installed which provides this information. This EULA incorporates the 'unlimited use' aspect of the APL64 runtime component which includes the CSE feature.

Are there any other licensing issues with respect to the CSE?

Since the CSE exposes the Microsoft .Net and components which have been developed using .Net, APL64 programmers who incorporate the CSE technology in their application systems need to be aware of licensing requirements which may apply. For example:

- Commercial component developers make available numerous tools based on .Net. These components are often available in demonstration/evaluation versions or even community versions. Generally, a commercial component developer will have their own license agreement.
- Opensource components based on .Net are also readily available, e.g. from sourceforge.net or stackoverflow.com. These components are often covered by a license agreement requiring attribution and other disclosures.
- For APL64 programmers with Microsoft Visual Studio knowledge, custom .Net assemblies can be created an accessed in APL64 using the CSE. It may be important to provide appropriate license terms for these custom components also.

I don't know C#. Do I need to learn it? How can I learn it?

If the CSE will be used to load and interact with previously compiled .Net assemblies, simple and minimal C# skill is needed to benefit from using the CSE. If CSE scripts will be written using C# syntax to create .Net classes, additional knowledge of the C# .Net programming language is beneficial. The C# programming language and the Microsoft .Net are both implemented and documented well and have been extensively deployed to develop productive and amazing applications on the Windows workstation and server operating system platforms.

APL64 supports interfaces to many programming 'environments' and the CSE provides a robust interface to the C# and .Net environment for those who need it.

Many avenues are available for a programmer interested in learning C# and .Net including:

- Microsoft Developer Network
- MSDN Library
- MSDN Library C#
- Stackoverflow for C# Questions
- Apress Publications for C#
- Visual Studio 2015 Community
- APL2000 Consulting Services

Why does deployment of the CSE require multiple .Net Assemblies?

Certain Microsoft-provided .Net assemblies used by the CSE cannot be merged into a single assembly.

CSE Examples

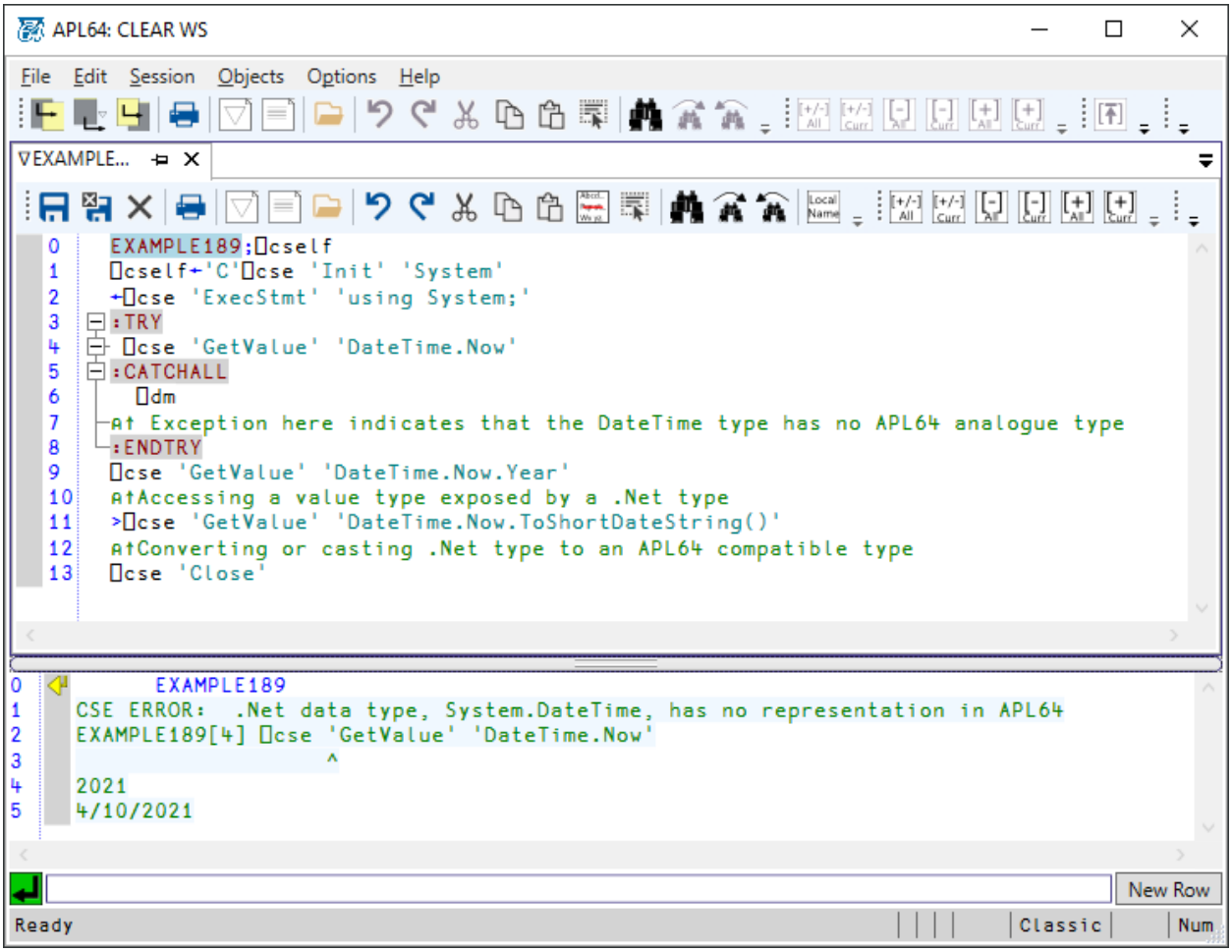
Most of the source code for the examples in the APL64 CSE manual are self-contained in the document. For these examples, copy the APL64 source code from the document into an instance of the APL64 developer version and run the copied function.

In some cases, additional files are needed to run the examples. These additional files are included in the 'CSE C# Script Engine Manual Example Files.zip' compressed file. Expand the content of this file to a convenient location on the target workstation. For the APL64 CSE examples which require additional files, the APL64 example function may need to be modified to reflect the path to the expanded zip file.

The examples are numbered as they are prepared and included in the CSE documentation. The numbering system for the examples is not necessarily consecutive or coordinated with the page numbers of this document. Some example numbers may not exist. Some examples do not have associated example workspaces. Some examples were created with different versions of APL64 or Microsoft Visual Studio, but these examples may also be created using the latest versions of APL64 and Microsoft Visual Studio. All CSE examples reflect the default values of CSE instance properties, unless otherwise stated in the CSE documentation or examples.

Example #189: Accessing a .Net datatype that has no representation in APL64
APL64 supports a limited set of 'value types', i.e. scalars and arrays of bool, characters, strings, Int32 and double. If the data type of a .Net variable, field or property value has no associated APL64 data type or if the APL64 interface cannot convert the .Net type to an APL64-compatible type, the CSE throws an exception. In this case a datatype conversion of the original .Net variable or access to its value type properties will be necessary to obtain the desired values in the APL64 session.

```
EXAMPLE189;□cself
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
:TRY
 □cse 'GetValue' 'DateTime.Now'
:CATCHALL
  □dm
⍝↑ Exception here indicates that the DateTime type has no APL64 analogue type
:ENDTRY
□cse 'GetValue' 'DateTime.Now.Year'
⍝↑Accessing a value type exposed by a .Net type
>□cse 'GetValue' 'DateTime.Now.ToShortDateString()'
⍝↑Converting or casting .Net type to an APL64 compatible type
□cse 'Close'
```



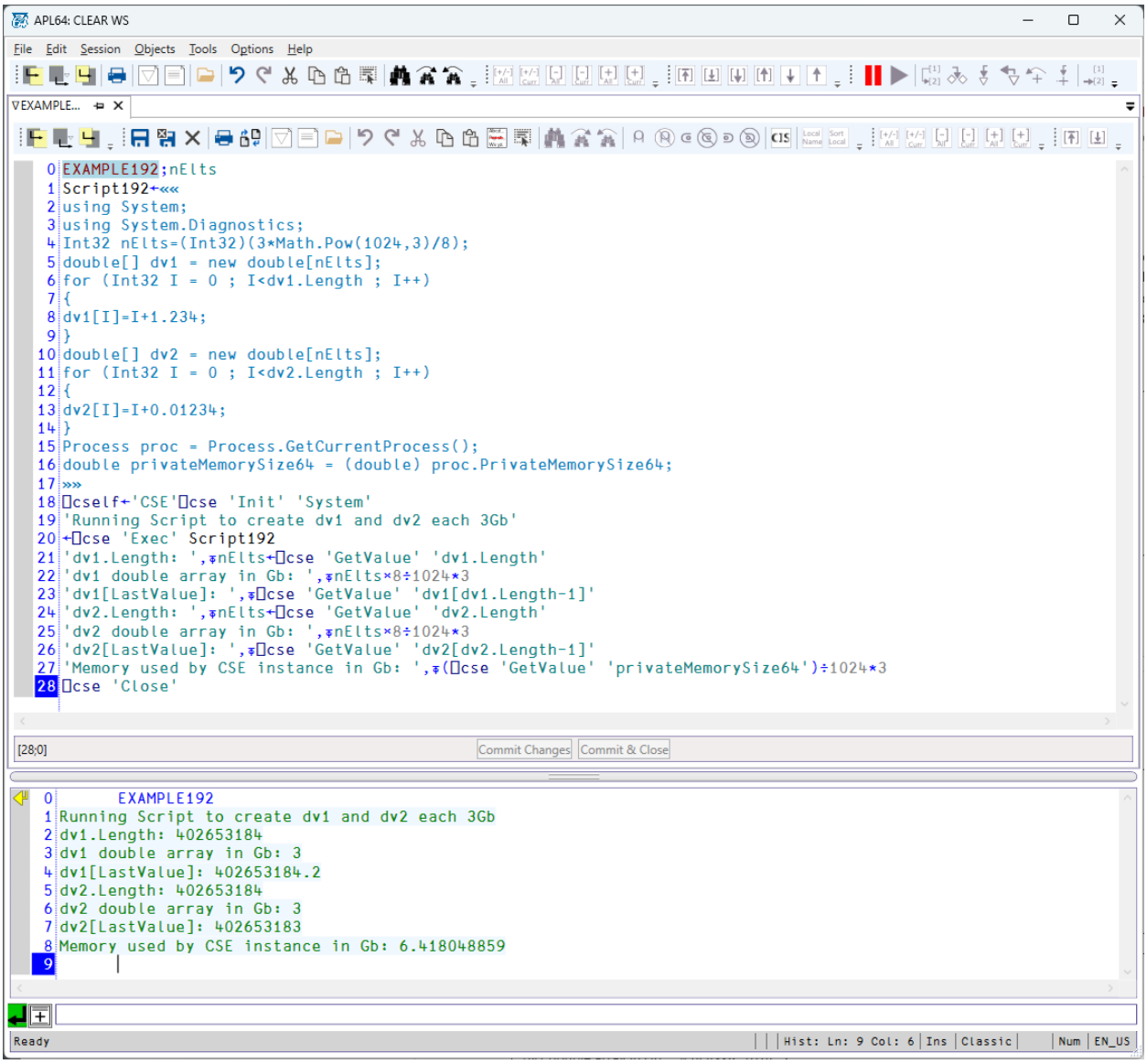
How much memory can the CSE use?

A CSE instance can use up to the available memory on the target workstation.

Example #192: In this example two .Net objects, each a 3Gb array of doubles, are created in a CSE instance using a Windows 7 X64 OS machine with 8Gb of memory. This illustrates that with a workstation with appropriate operating system and memory, the CSE can use memory above the Win32 memory limit and create .Net objects above the Win32 memory limit. The example uses arrays of double with 8 bytes/double of memory used plus a small amount of array structure overhead.

```
EXAMPLE192;nElts; Script192
Script192←««
using System;
using System.Diagnostics;
Int32 nElts=(Int32)(3*Math.Pow(1024,3)/8);
double[] dv1 = new double[nElts];
for (Int32 I = 0 ; I<dv1.Length ; I++)
{
dv1[I]=I+1.234;
```

```
}
double[] dv2 = new double[nElts];
for (Int32 I = 0 ; I<dv2.Length ; I++)
{
dv2[I]=I+0.01234;
}
Process proc = Process.GetCurrentProcess();
double privateMemorySize64 = (double) proc.PrivateMemorySize64;
»»
⎕cself←'CSE'⎕cse 'Init' 'System'
'Running Script to create dv1 and dv2 each 3Gb'
←⎕cse 'Exec' Script192
'dv1.Length: ', ⍕ nElts←⎕cse 'GetValue' 'dv1.Length'
'dv1 double array in Gb: ', ⍕ nElts×8÷1024*3
'dv1[LastValue]: ', ⍕ ⎕cse 'GetValue' 'dv1[dv1.Length-1]'
'dv2.Length: ', ⍕ nElts←⎕cse 'GetValue' 'dv2.Length'
'dv2 double array in Gb: ', ⍕ nElts×8÷1024*3
'dv2[LastValue]: ', ⍕ ⎕cse 'GetValue' 'dv2[dv2.Length-1]'
'Memory used by CSE instance in Gb: ', ⍕ (⎕cse 'GetValue' 'privateMemorySize64')÷1024*3
⎕cse 'Close'
```

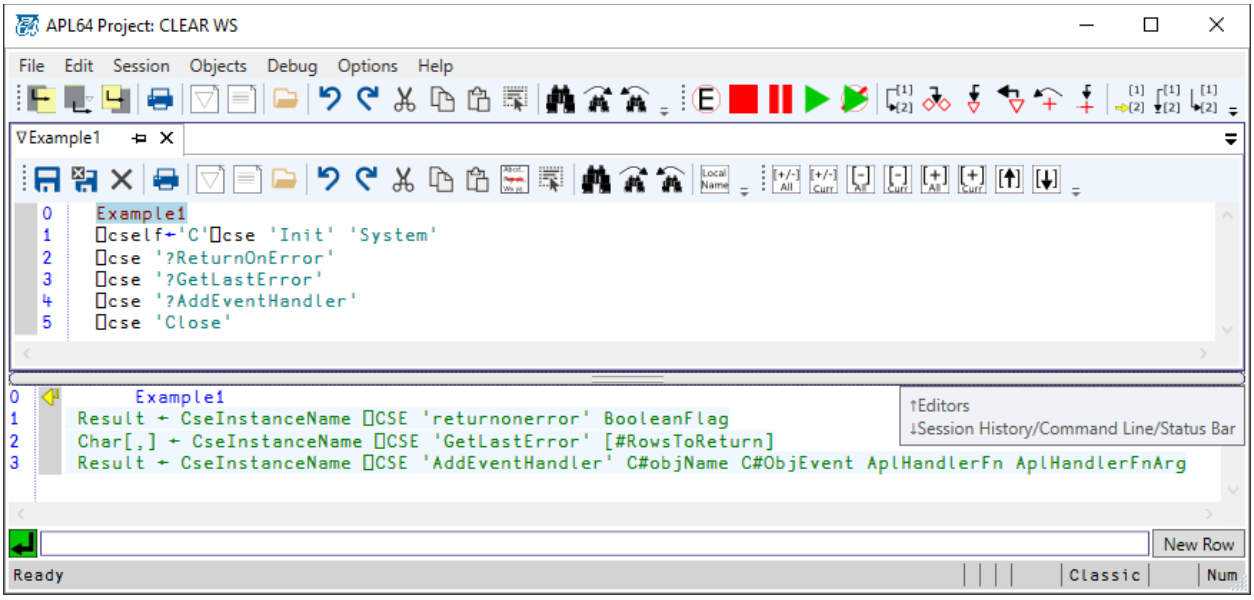


Is there a quick summary of an element of the CSE object model?

Yes. The CSE supports brief 'help strings' which can be accessed by prefixing the CSE object model element name with a question mark ("?"). Because a CSE 'help string' is intended to be a summary, some options for that CSE method or property may not be disclosed in the 'help string'. Refer to the CSE manual (this document) for complete documentation of the CSE object model.

Example #1:

```
Example1
□cself←'C'□cse 'Init' 'System'
□cse '?ReturnOnError'
□cse '?GetLastError'
□cse '?AddEventHandler'
□cse 'Close'
```

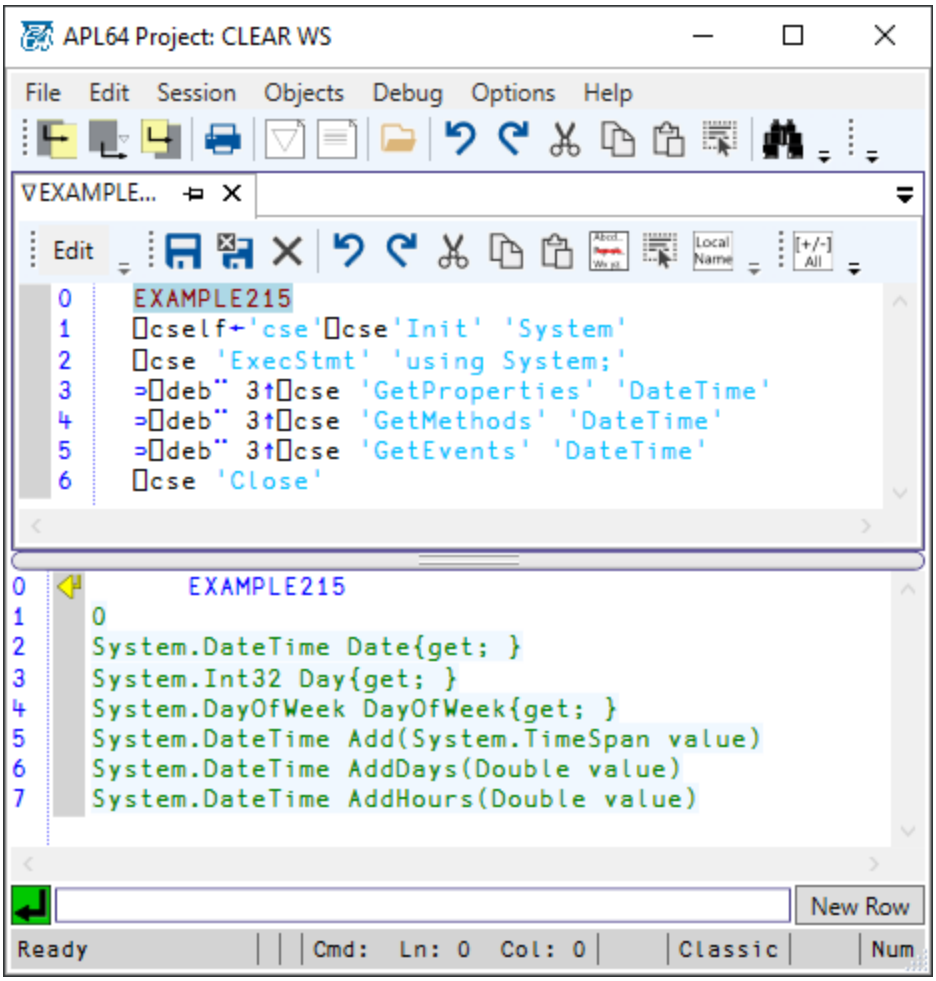


What is a good way to browse the object model of .Net types?

Use the CSE ‘GetProperties’, ‘GetMethods’ and ‘GetEvents’ methods to investigate the object model of .Net objects, classes and events.

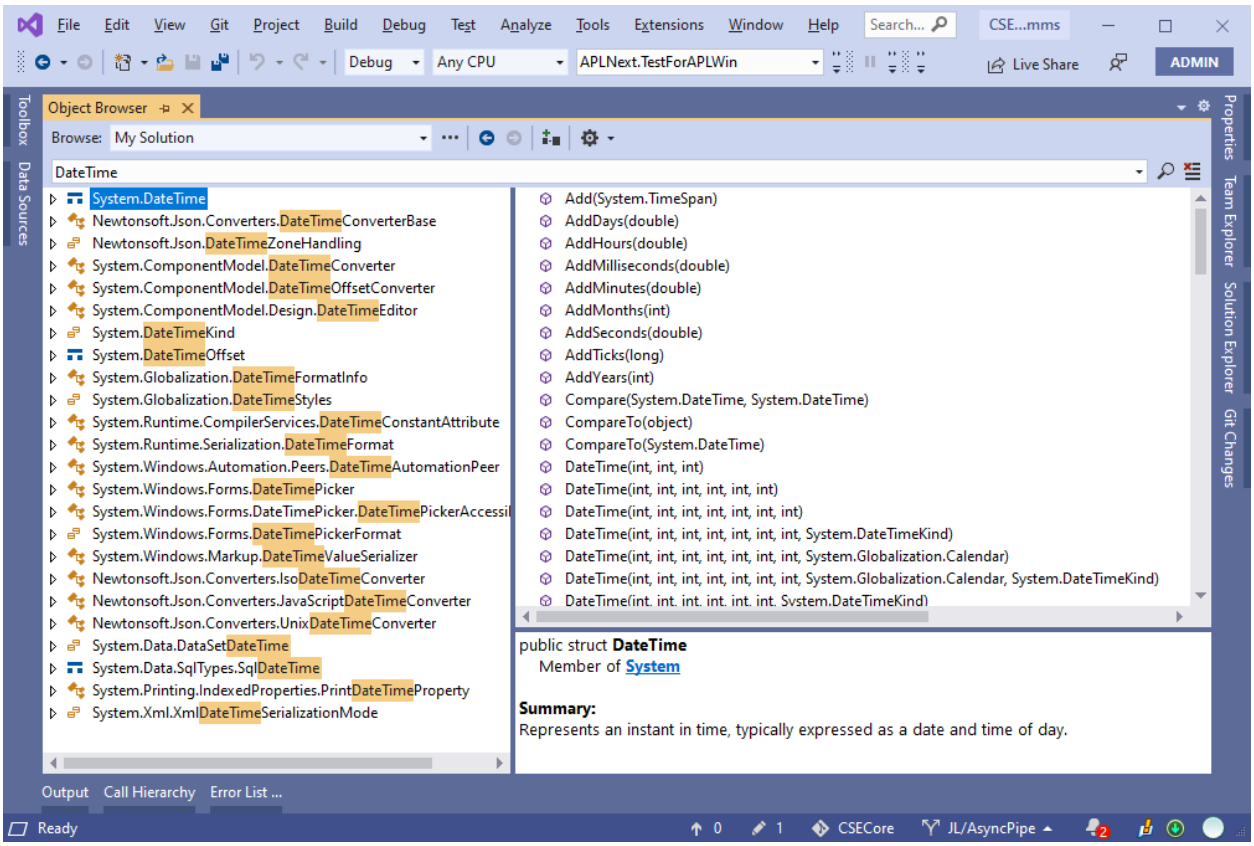
Example #215

```
EXAMPLE215
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⊃⎕deb¨ 3↑⎕cse 'GetProperties' 'DateTime'
⊃⎕deb¨ 3↑⎕cse 'GetMethods' 'DateTime'
⊃⎕deb¨ 3↑⎕cse 'GetEvents' 'DateTime'
⎕cse 'Close'
```



Alternative using Visual Studio:
The 'Object Browser' included in Visual Studio 2019 Community Version is a no-cost way to browse the object model (methods, properties and events) of classes (types) in .Net. For more detailed examination of .Net types try these tools.

For example, after installing Visual Studio 2019 Community Version, use the 'View > Object Browser' menu item and search for DateTime to view the object model for the 'DateTime' type, search for DateTime:



Are CSE instances persisted between APL64 sessions?

No. CSE instances end when an APL64 session ends. Thus CSE system variable values are not persisted between APL64 sessions. Note however that certain .Net objects may be created via the CSE during an APL64 session which will persist after the APL64 session ends. Such .Net object types should be 'disposed' prior to ending an APL64 session which created those objects. For example:

- An ADO.Net database connection left open when the APL64 session which created it closes
- A System.IO file access restriction which is not resolved before the APL64 session which created it closes
- An ActiveX object instance created but not closed before the APL64 session which created it closes

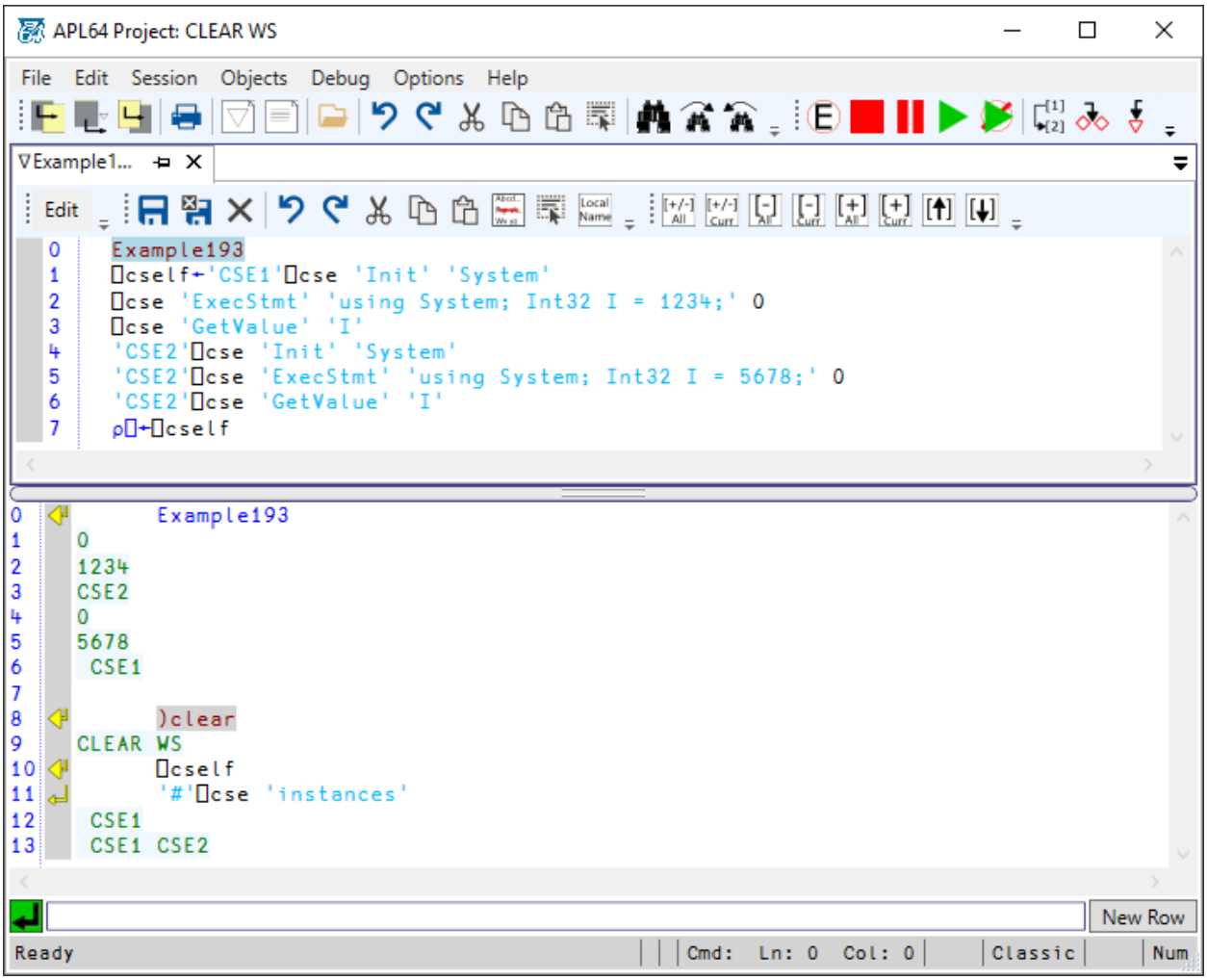
Does)Clear Close a CSE Instance?

No. When)Clear is used, □cself and pre-existing CSE instances are preserved.

Example #193:

```
Example193
□cself←'CSE1'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System; Int32 I = 1234;'
□cse 'GetValue' 'I'
'CSE2'□cse 'Init' 'System'
'CSE2'□cse 'ExecStmt' 'using System; Int32 I = 5678;'
'CSE2'□cse 'GetValue' 'I'
```

```
ρ□←□cself
```



How many CSE instances may be created in an APL64 session?

The number of instances of the CSE which may be created during an APL64 session is limited by the memory available to the target workstation considering all applications and services running on that machine.

CSE Performance Tips

Create the minimum required number of CSE instances, re-use them throughout the APL64 session and close when no longer needed.

Dispose of large .Net objects before using □cse 'Close'. Before the CSE 'Close' method is used, it is recommended that any large .Net objects created in the associated instance of the C# script engine be disposed by the programmer before using the CSE 'Close' method. For most .Net objects disposing of them can be done by setting them equal to null.

Do not create and close CSE instances in a loop. For example, it is not recommended to create and close instances of the CSE inside of an APL64 loop. Create a CSE instance before the APL64 loop processing commences and use that CSE instance withing the loop.

Important APL64 and C# Comparisons

.Net and C# Names are Case Sensitive

Object names in .Net, including the C# language are case sensitive.

C Sharp objects are strongly-typed

C# objects are strongly-typed so that once the data type of a named C# object is established in an executing C# .Net assembly, that named object's type cannot normally be changed. The C# Roslyn compiler is analogous to the APL64 interpreter, so at the expense of performance, it is possible to assign different data type values to the same variable name. For example, reassigning the same variable name to different C# data types is more expensive than using separate variable names:

```
var X = 0;
var X= "ABCD";
```

compared to

```
var X = 0;
var Y= "ABCD";
```

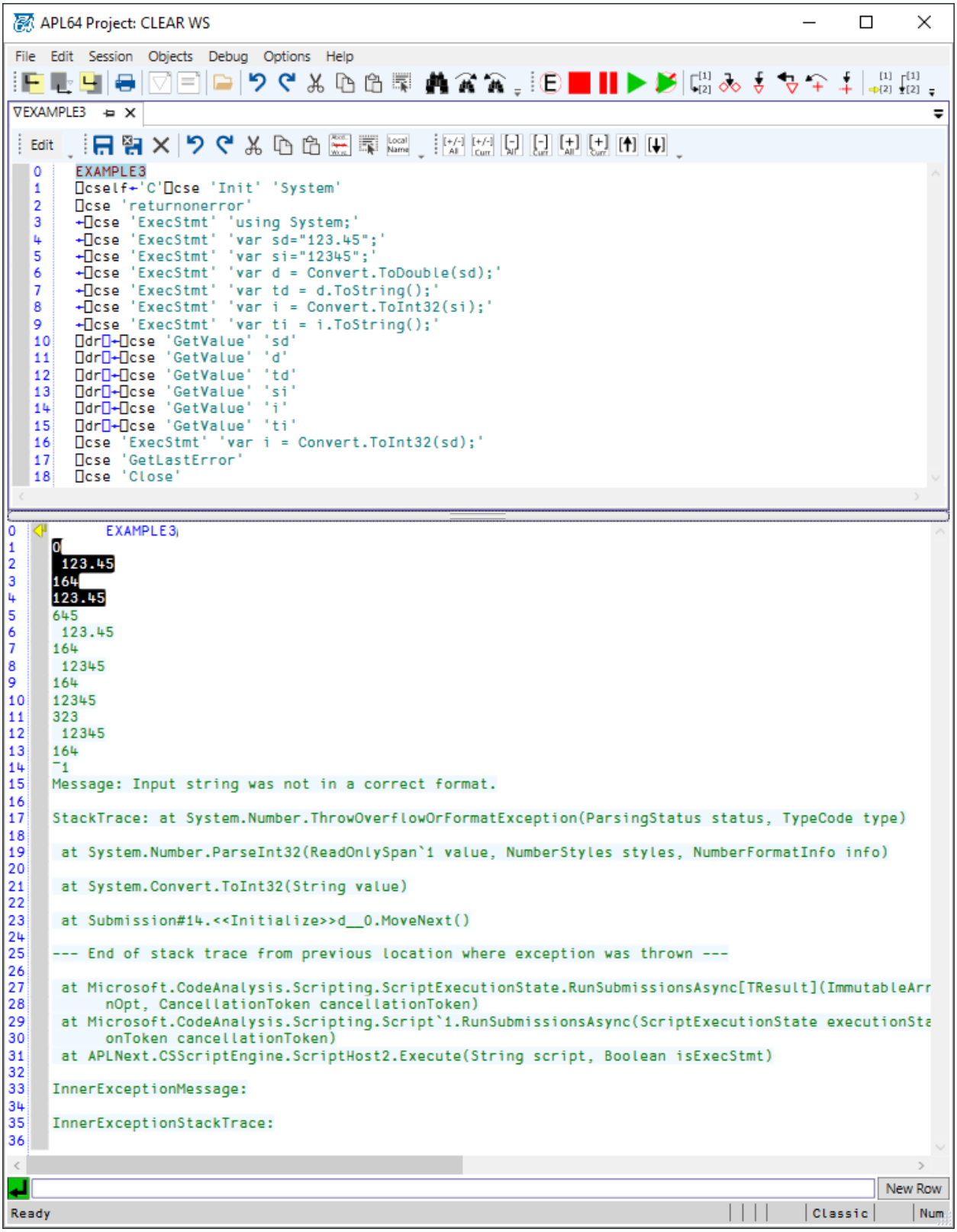
It is possible to create new C# objects of a different type from existing C# objects using conversion or casting. The data type of a C# object can be established implicitly by the C# compiler using the 'var' type in which case the data type of that C# object is determined at runtime.

Example #3:

In this example explicit C# conversions are used to create new C# objects of different types. C# does not support all possible explicit conversions, as in this example: Convert.ToInt32(sd) where the argument, sd, is not an integer. Some explicit C# conversions can result in a loss of precision.

```
EXAMPLE3
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
←□cse 'ExecStmt' 'var sd="123.45";'
←□cse 'ExecStmt' 'var si="12345";'
←□cse 'ExecStmt' 'var d = Convert.ToDouble(sd);'
←□cse 'ExecStmt' 'var td = d.ToString();'
←□cse 'ExecStmt' 'var i = Convert.ToInt32(si);'
←□cse 'ExecStmt' 'var ti = i.ToString();'
□dr□←□cse 'GetValue' 'sd'
□dr□←□cse 'GetValue' 'd'
□dr□←□cse 'GetValue' 'td'
□dr□←□cse 'GetValue' 'si'
□dr□←□cse 'GetValue' 'i'
```

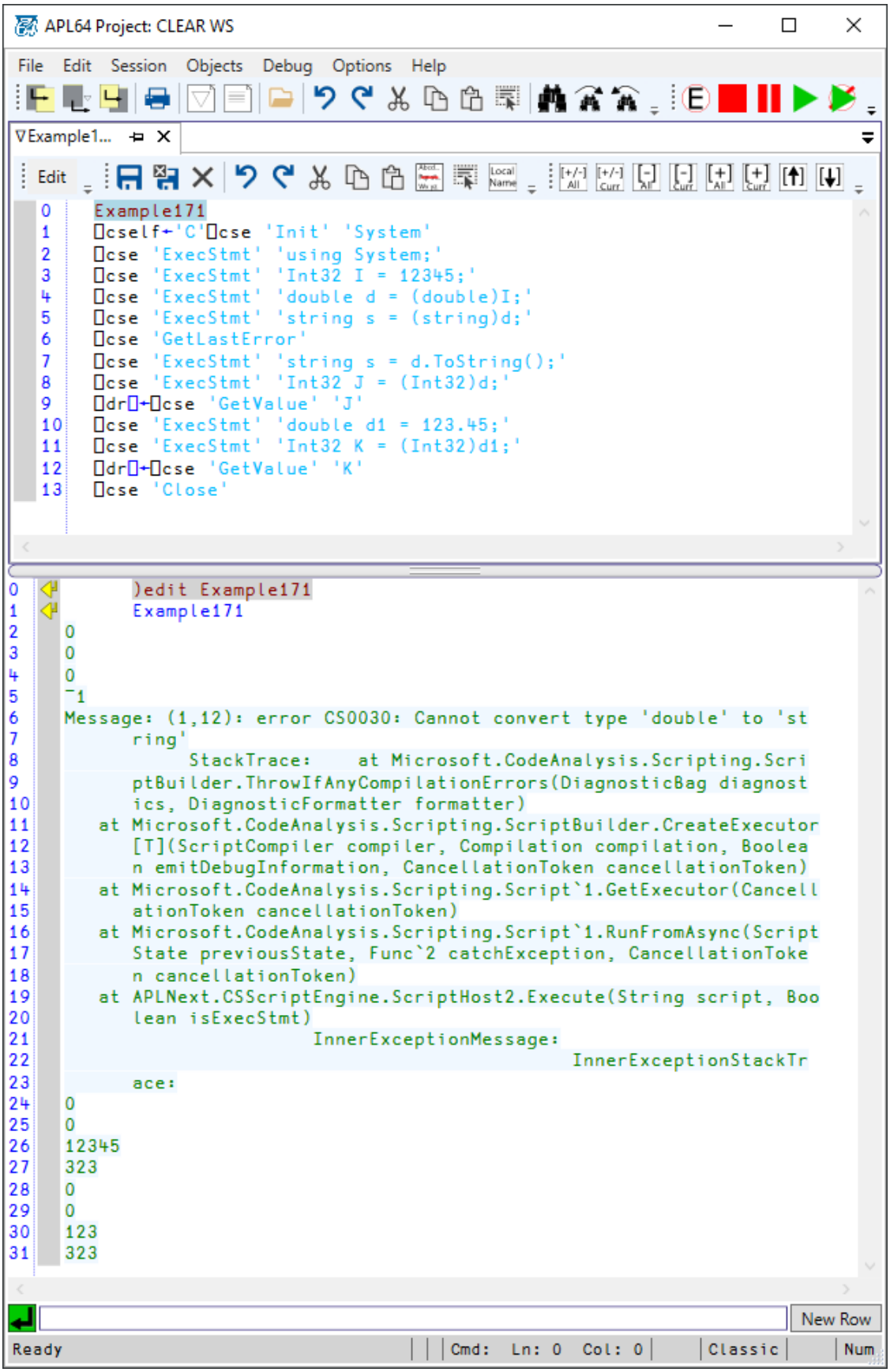
```
⎕dr⎕←⎕cse 'GetValue' 'ti'
⎕cse 'ExecStmt' 'var i = Convert.ToInt32(sd);'
⎕cse 'GetLastError'
←⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
∇EXAMPLE3
Edit
0 EXAMPLE3
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'returnonerror'
3 ⎕cse 'ExecStmt' 'using System;'
4 ⎕cse 'ExecStmt' 'var sd="123.45";'
5 ⎕cse 'ExecStmt' 'var si="12345";'
6 ⎕cse 'ExecStmt' 'var d = Convert.ToDouble(sd);'
7 ⎕cse 'ExecStmt' 'var td = d.ToString();'
8 ⎕cse 'ExecStmt' 'var i = Convert.ToInt32(si);'
9 ⎕cse 'ExecStmt' 'var ti = i.ToString();'
10 ⎕dr⎕cse 'GetValue' 'sd'
11 ⎕dr⎕cse 'GetValue' 'd'
12 ⎕dr⎕cse 'GetValue' 'td'
13 ⎕dr⎕cse 'GetValue' 'si'
14 ⎕dr⎕cse 'GetValue' 'i'
15 ⎕dr⎕cse 'GetValue' 'ti'
16 ⎕cse 'ExecStmt' 'var i = Convert.ToInt32(sd);'
17 ⎕cse 'GetLastError'
18 ⎕cse 'Close'
EXAMPLE3
0
123.45
164
123.45
645
123.45
164
12345
164
12345
323
12345
164
¯1
Message: Input string was not in a correct format.
StackTrace: at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)
at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)
at System.Convert.ToInt32(String value)
at Submission#14.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAsync[TResult](ImmutableArr
nOpt, CancellationToken cancellationToken)
at Microsoft.CodeAnalysis.Scripting.Script`1.RunSubmissionsAsync(ScriptExecutionState executionSta
onToken cancellationToken)
at APLNext.CSScriptEngine.ScriptHost2.Execute(String script, Boolean isExecStmt)
InnerExceptionMessage:
InnerExceptionStackTrace:
New Row
Ready
Classic
Num

Example #171:

In this example C# ‘casting’ is used to create new C# objects of different types. C# does not support all casts, as in this example (string)d, where d is a double, whereas d.ToString() is available. Some C# casts can result in a loss of precision.

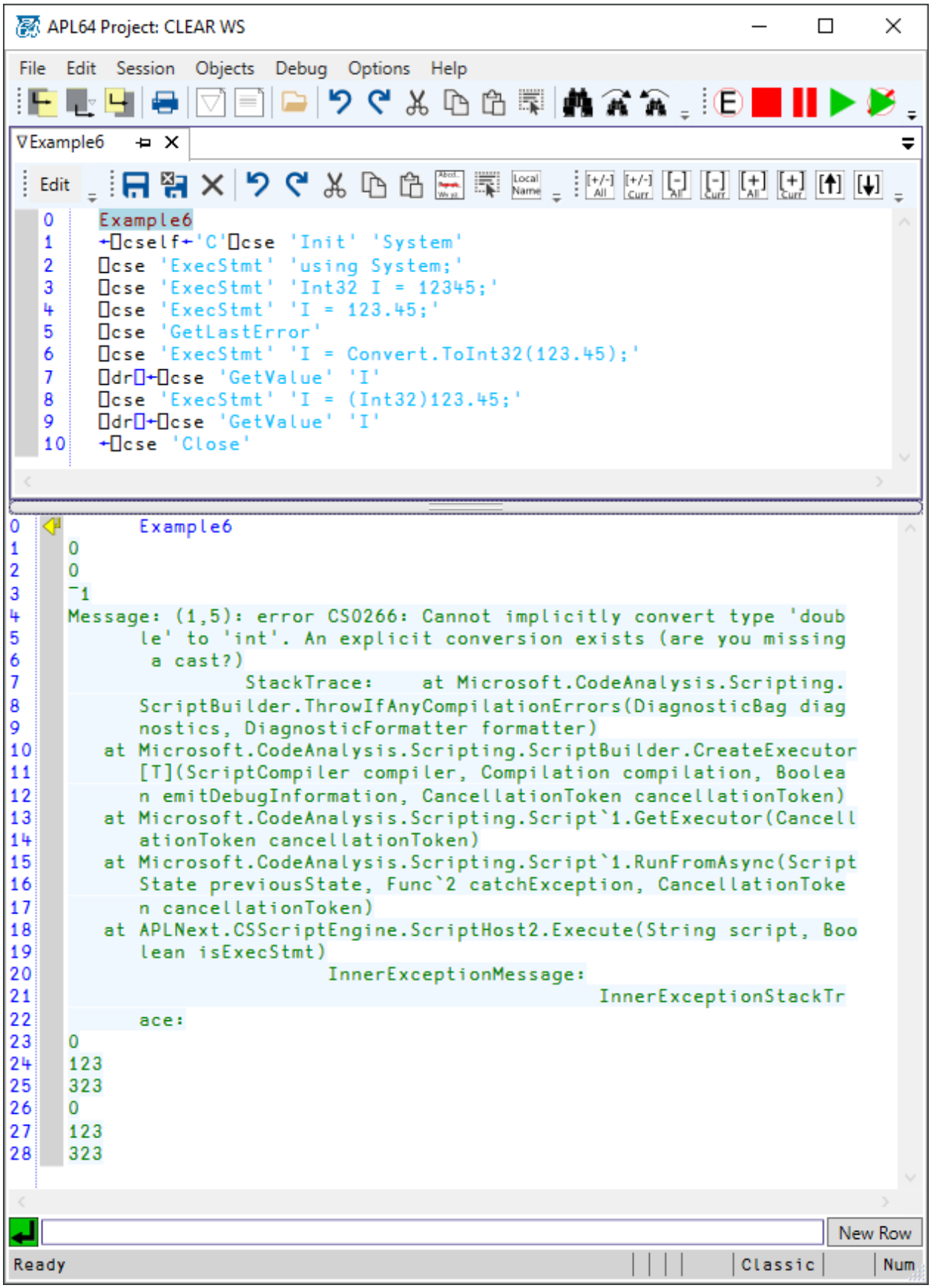
```
Example171
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'Int32 I = 12345;'
□cse 'ExecStmt' 'double d = (double)I;'
□cse 'ExecStmt' 'string s = (string)d;'
□cse 'GetLastError'
□cse 'ExecStmt' 'string s = d.ToString();'
□cse 'ExecStmt' 'Int32 J = (Int32)d;'
□dr□←□cse 'GetValue' 'J'
□cse 'ExecStmt' 'double d1 = 123.45;'
□cse 'ExecStmt' 'Int32 K = (Int32)d1;'
□dr□←□cse 'GetValue' 'K'
□cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
Example1...
Edit
0 Example171
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'ExecStmt' 'using System;'
3 ⎕cse 'ExecStmt' 'Int32 I = 12345;'
4 ⎕cse 'ExecStmt' 'double d = (double)I;'
5 ⎕cse 'ExecStmt' 'string s = (string)d;'
6 ⎕cse 'GetLastError'
7 ⎕cse 'ExecStmt' 'string s = d.ToString();'
8 ⎕cse 'ExecStmt' 'Int32 J = (Int32)d;'
9 ⎕dr⎕←⎕cse 'GetValue' 'J'
10 ⎕cse 'ExecStmt' 'double d1 = 123.45;'
11 ⎕cse 'ExecStmt' 'Int32 K = (Int32)d1;'
12 ⎕dr⎕←⎕cse 'GetValue' 'K'
13 ⎕cse 'Close'
0)edit Example171
1 Example171
2 0
3 0
4 0
5 ¯1
6 Message: (1,12): error CS0030: Cannot convert type 'double' to 'st
7 ring'
8 StackTrace: at Microsoft.CodeAnalysis.Scripting.Scri
9 ptBuilder.ThrowIfAnyCompilationErrors(DiagnosticBag diagnost
10 ics, DiagnosticFormatter formatter)
11 at Microsoft.CodeAnalysis.Scripting.ScriptBuilder.CreateExecutor
12 [T](ScriptCompiler compiler, Compilation compilation, Boolea
13 n emitDebugInformation, CancellationToken cancellationToken)
14 at Microsoft.CodeAnalysis.Scripting.Script`1.GetExecutor(Cancell
15 ationToken cancellationToken)
16 at Microsoft.CodeAnalysis.Scripting.Script`1.RunFromAsync(Script
17 State previousState, Func`2 catchException, CancellationToke
18 n cancellationToken)
19 at APLNext.CSScriptEngine.ScriptHost2.Execute(String script, Boo
20 lean isExecStmt)
21 InnerExceptionMessage:
22 InnerExceptionStackTr
23 ace:
24 0
25 0
26 12345
27 323
28 0
29 0
30 123
31 323
New Row
Ready
Cmd: Ln: 0 Col: 0
Classic
Num

Example #6: In this example:

- Assign an integer value, 12345, to a strongly-type variable, I
- Attempt to set the value of a variable typed as integer to a double value, 123.45.
- Use a .Net conversion from 123.45 to an integer with loss of precision
- Use a .Net cast to convert 123.45 to an integer with loss of precision

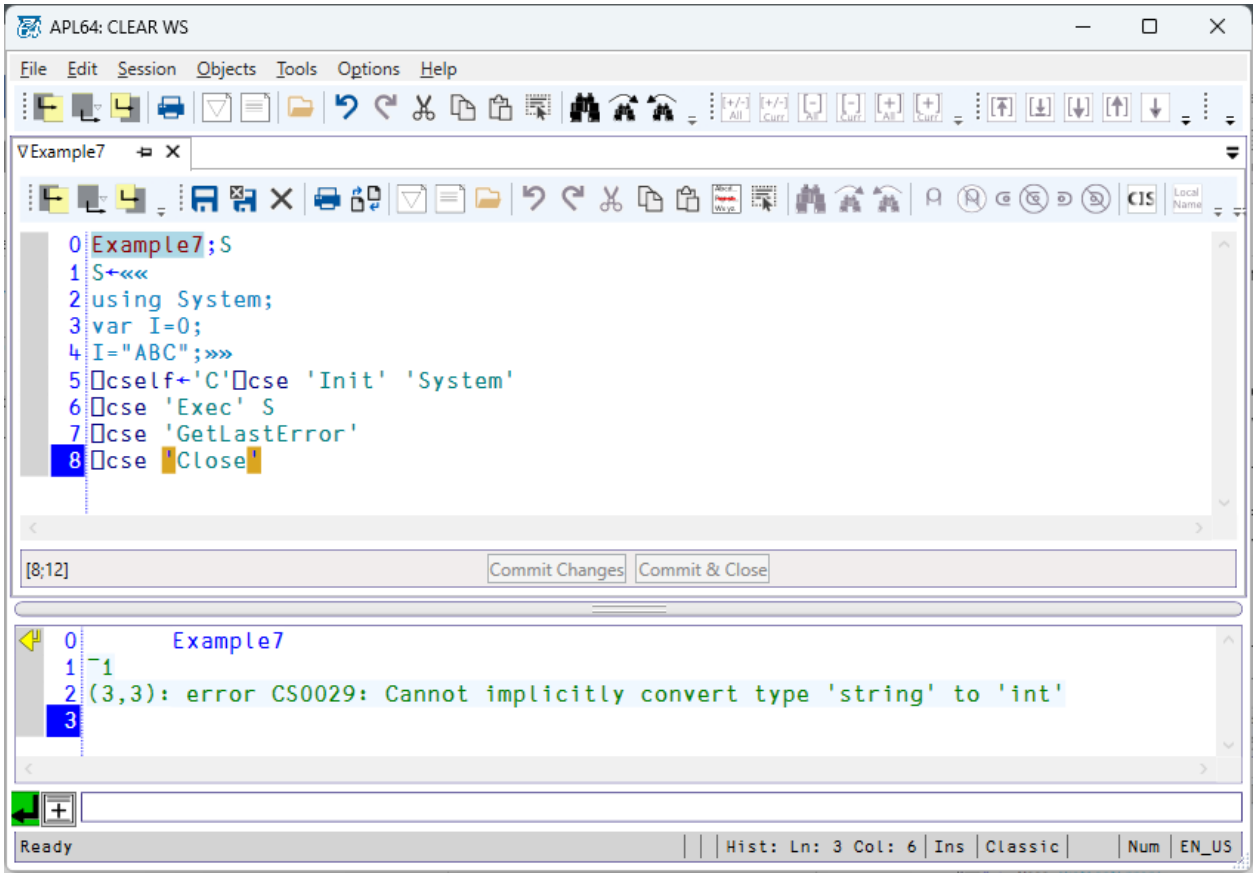
```
Example6
←⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'Int32 I = 12345;'
⎕cse 'ExecStmt' 'I = 123.45;'
⎕cse 'GetLastError'
⎕cse 'ExecStmt' 'I = Convert.ToInt32(123.45);'
⎕dr⎕←⎕cse 'GetValue' 'I'
⎕cse 'ExecStmt' 'I = (Int32)123.45;'
⎕dr⎕←⎕cse 'GetValue' 'I'
←⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
Example6
Edit
0 Example6
1 ←⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'ExecStmt' 'using System;'
3 ⎕cse 'ExecStmt' 'Int32 I = 12345;'
4 ⎕cse 'ExecStmt' 'I = 123.45;'
5 ⎕cse 'GetLastError'
6 ⎕cse 'ExecStmt' 'I = Convert.ToInt32(123.45);'
7 ⎕dr⎕←⎕cse 'GetValue' 'I'
8 ⎕cse 'ExecStmt' 'I = (Int32)123.45;'
9 ⎕dr⎕←⎕cse 'GetValue' 'I'
10 ←⎕cse 'Close'
0 Example6
1 0
2 0
3 ¯1
4 Message: (1,5): error CS0266: Cannot implicitly convert type 'doub
5 le' to 'int'. An explicit conversion exists (are you missing
6 a cast?)
7 StackTrace: at Microsoft.CodeAnalysis.Scripting.
8 ScriptBuilder.ThrowIfAnyCompilationErrors(DiagnosticBag diag
9 nostics, DiagnosticFormatter formatter)
10 at Microsoft.CodeAnalysis.Scripting.ScriptBuilder.CreateExecutor
11 [T](ScriptCompiler compiler, Compilation compilation, Boolea
12 n emitDebugInformation, CancellationToken cancellationToken)
13 at Microsoft.CodeAnalysis.Scripting.Script`1.GetExecutor(Cancell
14 ationToken cancellationToken)
15 at Microsoft.CodeAnalysis.Scripting.Script`1.RunFromAsync(Script
16 State previousState, Func`2 catchException, CancellationToke
17 n cancellationToken)
18 at APLNext.CSScriptEngine.ScriptHost2.Execute(String script, Boo
19 lean isExecStmt)
20 InnerExceptionMessage:
21 InnerExceptionStackTr
22 ace:
23 0
24 123
25 323
26 0
27 123
28 323
New Row
Ready
Classic
Num

The CSE supports dynamic typing for C# variables in some circumstances. In the CSE it is possible to redefine a named C# object's type when such redefinition is not part of method, class or property definition and if the redefinition is performed in separate CSE scripts or via individual CSE 'ExecStmt' method calls and if the variable's type is redefined when its value is changed.

Example #7:
The C# debugger/compiler will not permit the following syntax if it is included in a single CSE script because the debugger/compiler will analyze the entire script to see if it is valid at debug time.

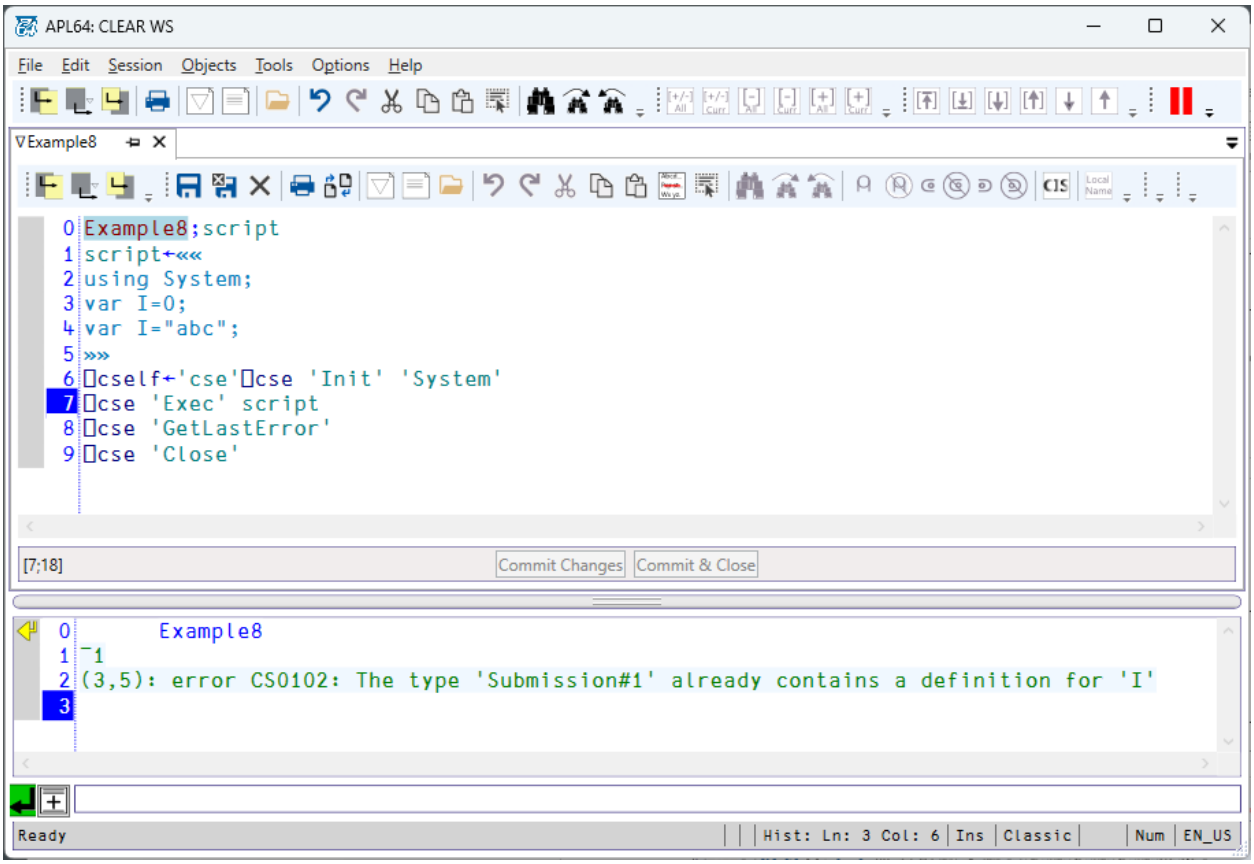
```
Example7;S
S←««
using System;
var I=0;
I="ABC";»»
□cself←'C'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'GetLastError'
□cse 'Close'
```



Example #8

The C# debugger/compiler will not permit the following syntax if it is included in a single CSE script because the debugger/compiler will analyze the entire script to see if it is valid at run time:

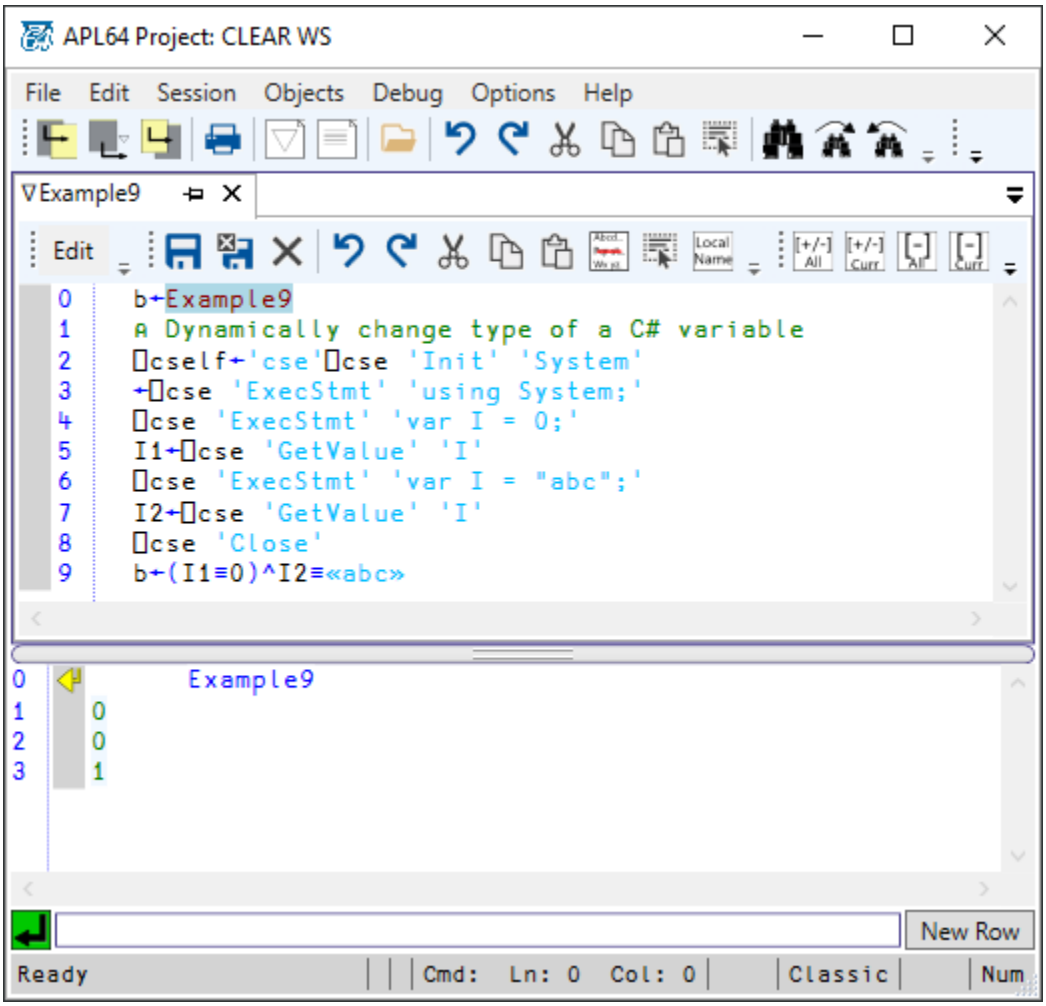
```
Example8;script
script←««
using System;
var I=0;
var I="abc";
»»
⍞←cself←'cse'⍞←cse 'Init' 'System'
⍞←cse 'Exec' script
⍞←cse 'GetLastError'
⍞←cse 'Close'
```



Example #9:

In this example two independent CSE 'ExecStmt' method calls using the C# 'var' keyword are executed to dynamically modify the data type of the C# variable 'I'. Because the change of data type for the C# variable 'I' does not occur within a single script, method, class or property definition and because the variable's type is redefined (via the var I = … syntax), this action is permitted by the C# debugger. Each use of the CSE ExecStmt method is effectively executing a separate C# script.

```
b←Example9
⍝ Dynamically change type of a C# variable
□cself←'cse'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'var I = 0;'
I1←□cse 'GetValue' 'I'
□cse 'ExecStmt' 'var I = "abc";'
I2←□cse 'GetValue' 'I'
□cse 'Close'
b←(I1≡0)^I2≡«abc»
```

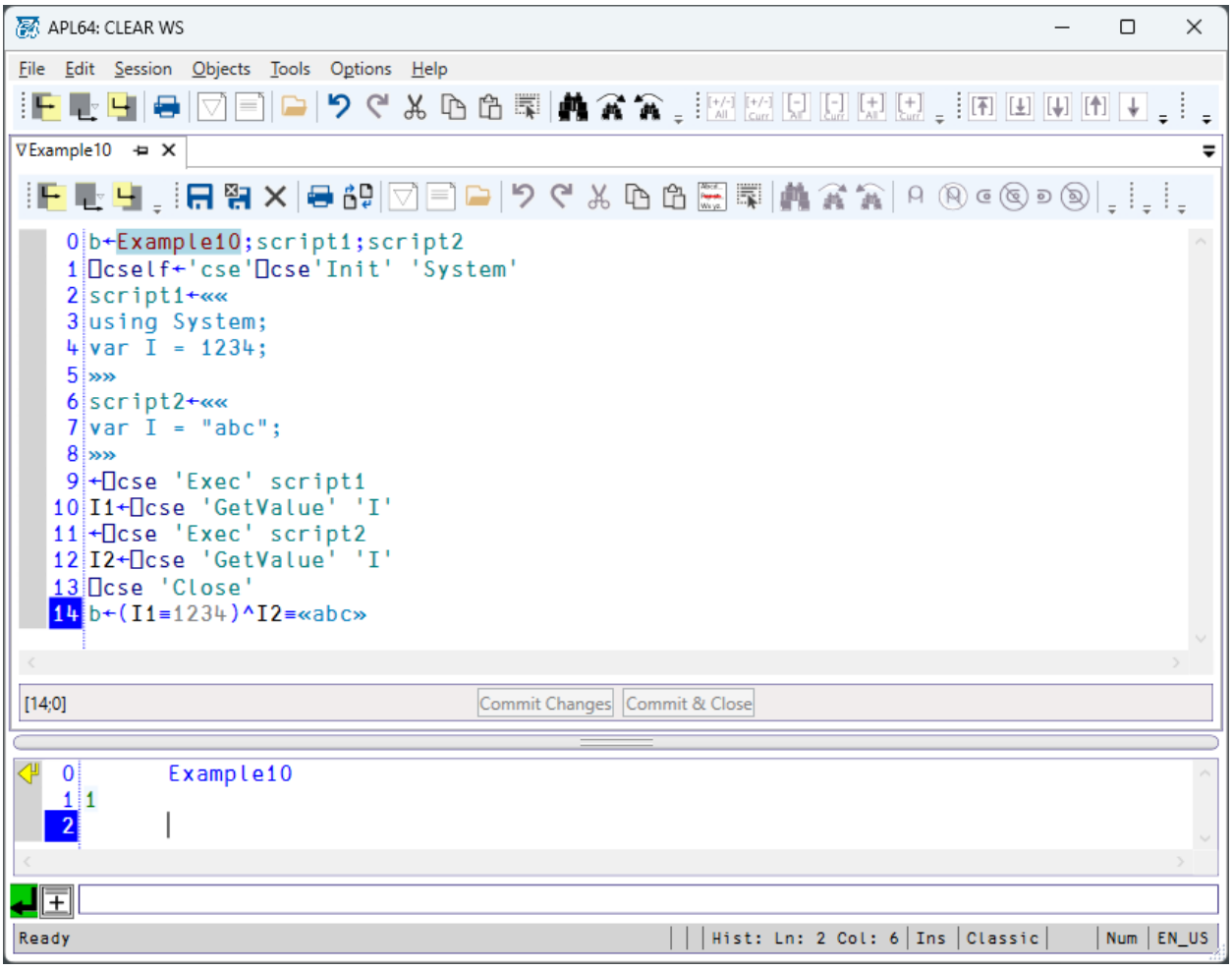


Example #10:

In this example two independent CSE scripts are executed using explicit .Net type names to dynamically modify the data type of the C# variable 'I':

```
b←Example10;script1;script2
□cself←'cse'□cse'Init' 'System'
```

```
script1←««
using System;
var I = 1234;
»»
script2←««
var I = "abc";
»»
←⎕cse 'Exec' script1
I1←⎕cse 'GetValue' 'I'
←⎕cse 'Exec' script2
I2←⎕cse 'GetValue' 'I'
⎕cse 'Close'
b←(I1≡1234)^I2≡«abc»
```



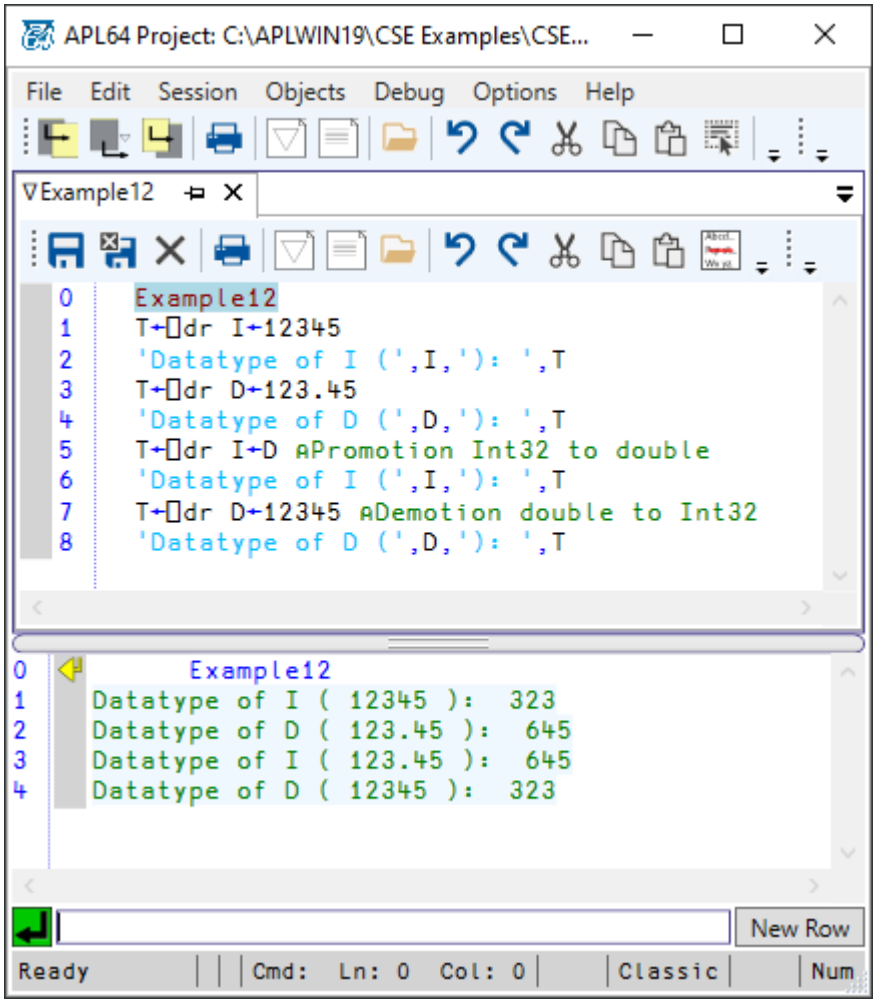
APL64 incorporates promotion and demotion of data types

APL64 promotes or demotes certain data types to a conformable data type when necessary and possible. In C# promotion is implemented for certain methods but demotion is not when a loss of data precision is possible. See the CSE 'SetValue' method documentation for more information and options.

Example #12:

This example illustrates promotion and demotion of data types in APL64.

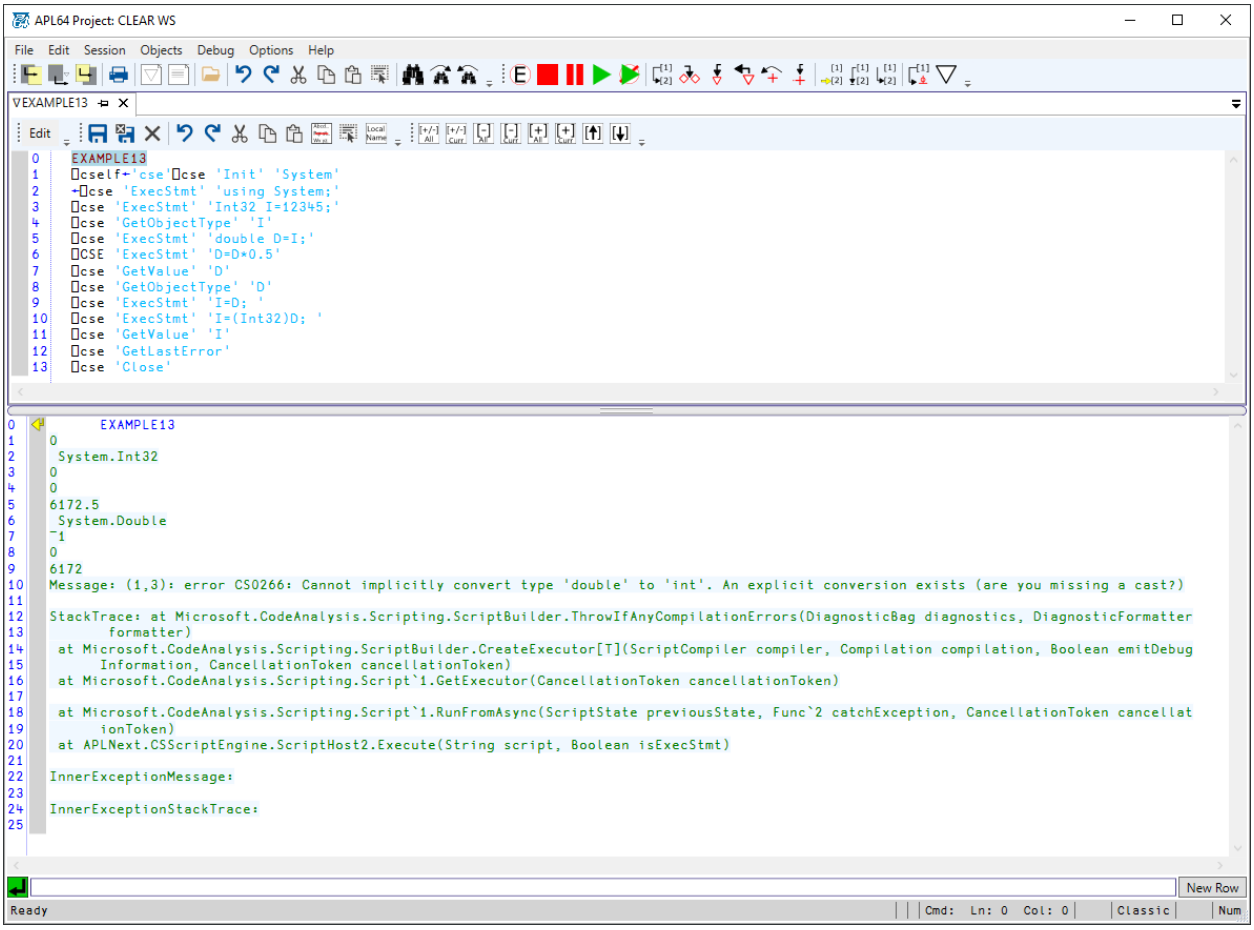
```
Example12
T←⎕dr I←12345
'Datatype of I (',I,'): ',T
T←⎕dr D←123.45
'Datatype of D (',D,'): ',T
T←⎕dr I←D ⍝Promotion Int32 to double
'Datatype of I (',I,'): ',T
T←⎕dr D←12345 ⍝Demotion double to Int32
'Datatype of D (',D,'): ',T
```



Example #13

In this example C# promotion is illustrated, but there is no implicit demotion in C#, so the attempt to set the value of the integer variable I to a double value fails to avoid a loss of precision. Using an explicit conversion and accepting the loss of precision may be used.

```
EXAMPLE13
 cself←'cse' cse 'Init' 'System'
←cse 'ExecStmt' 'using System;'
 cse 'ExecStmt' 'Int32 I=12345;'
 cse 'GetObjectType' 'I'
 cse 'ExecStmt' 'double D=I;'
 CSE 'ExecStmt' 'D=D*0.5'
 cse 'GetValue' 'D'
 cse 'GetObjectType' 'D'
 cse 'ExecStmt' 'I=D; '
 cse 'ExecStmt' 'I=(Int32)D;'
 cse 'GetValue' 'I'
 cse 'GetLastError'
 cse 'Close'
```



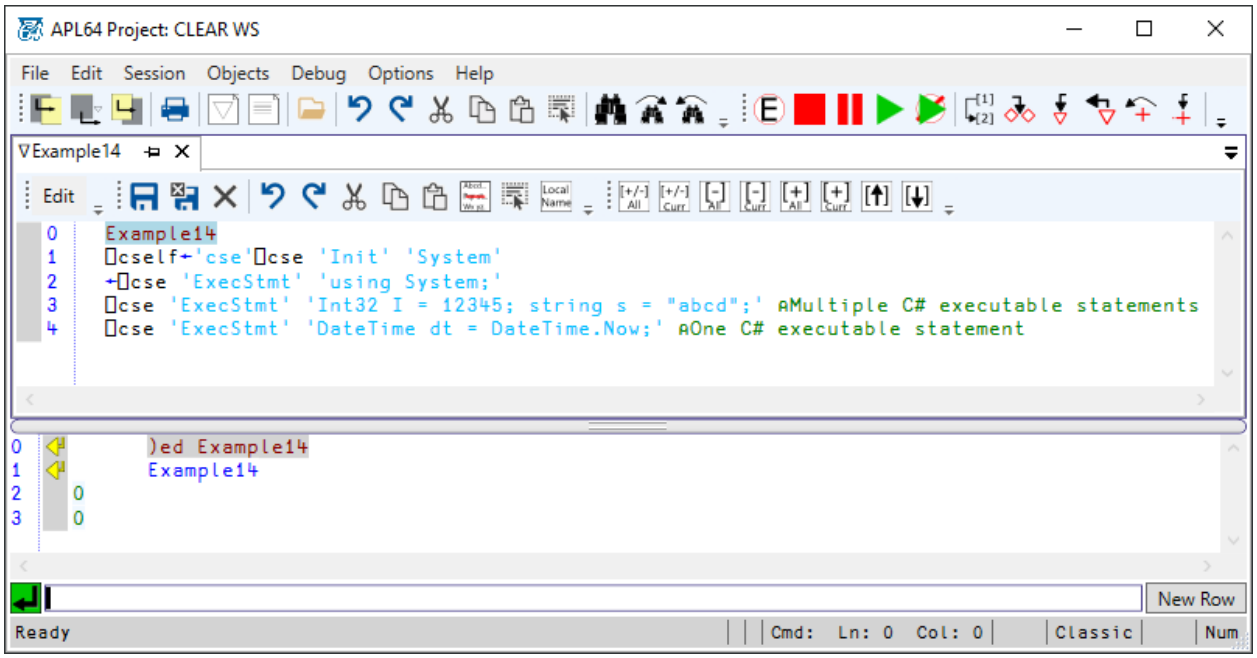
C Sharp Semi-colon statement separator

The statement separator in C# is the semi-colon (";"). It is used between multiple C# executable statements on a line of source code. In addition a 'line' of C# source code must end with the semi-colon statement separator, even if it contains only one C# statement, because in C# a line of source code may span more than one line. In APL64 the diamond ("⋄") is the statement separator and it is not required at the end of an APL64 executable statement.

Example #14:

For C# executable statements a semi-colon line separator is required.

```
Example14
⎕cself←'cse'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'Int32 I = 12345; string s = "abcd";' ⍝Multiple C# executable statements
⎕cse 'ExecStmt' 'DateTime dt = DateTime.Now;' ⍝One C# executable statement
```

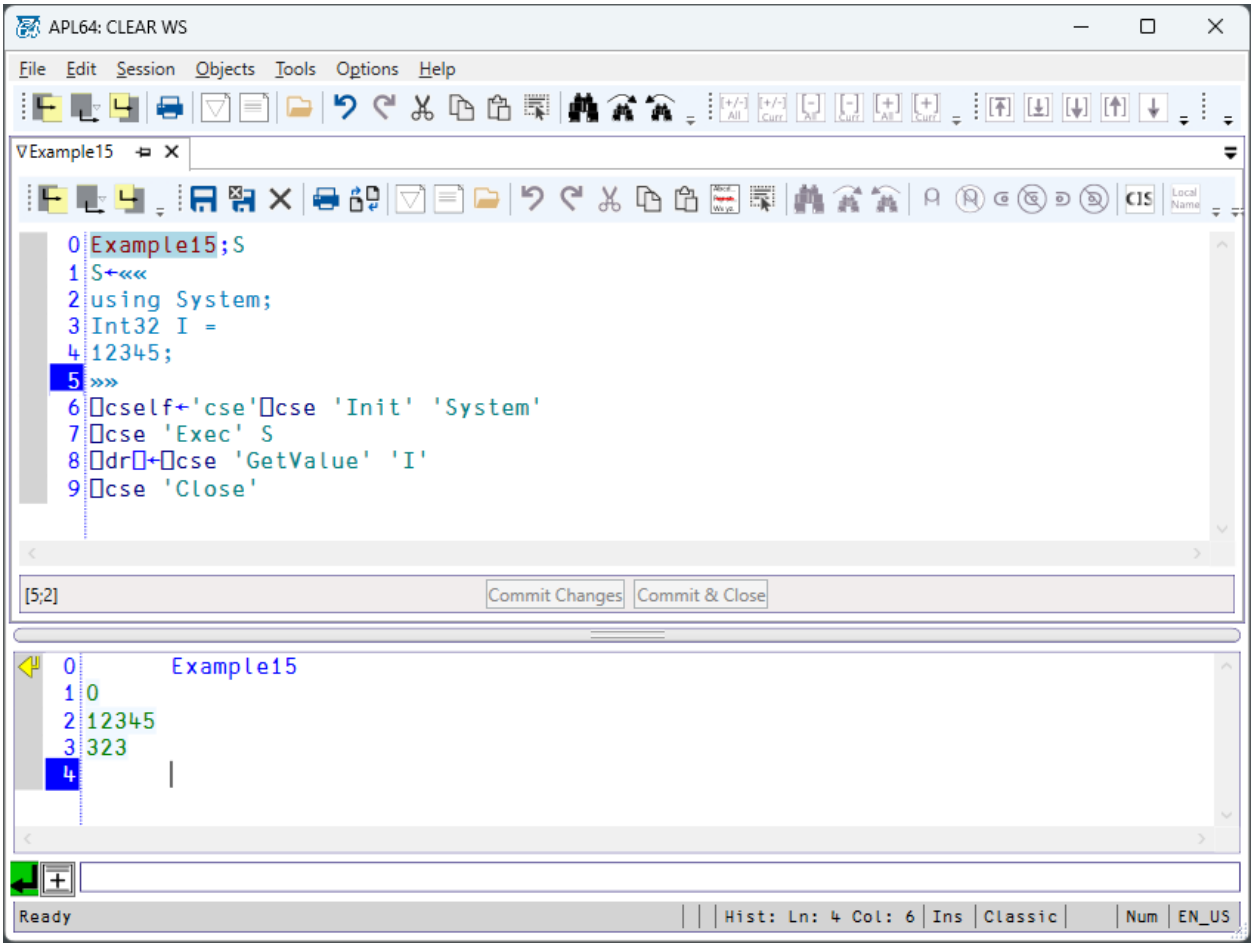


Example #15:

In this example a CSE script contains a C# executable statement which uses two lines of the script to define the C# variable 'I' and set its value.

```
Example15;S
S←⍬
using System;
Int32 I =
```

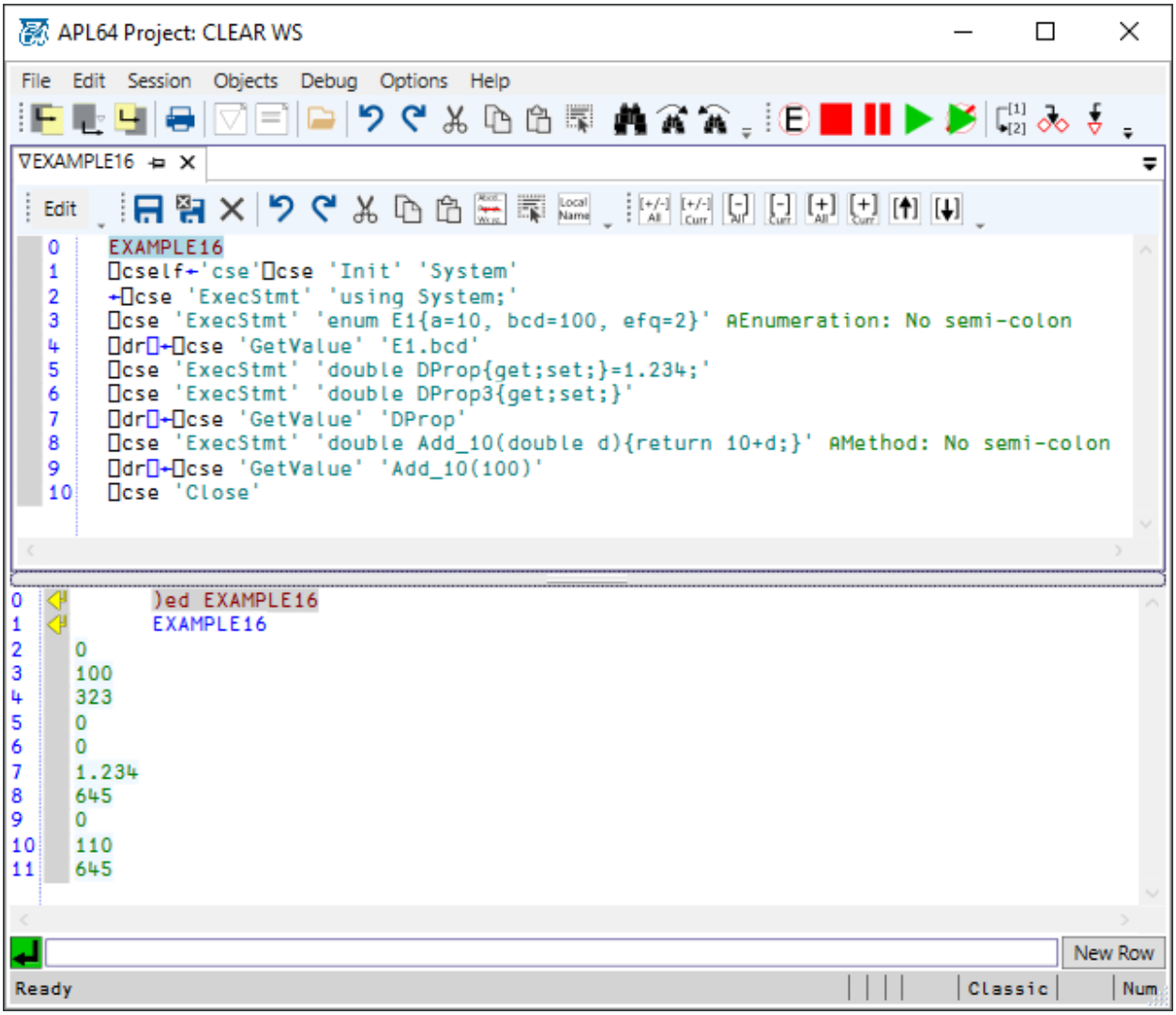
```
12345;
»»
⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'Exec' S
⎕dr⎕←⎕cse 'GetValue' 'I'
⎕cse 'Close'
```



Example #16: Certain C# syntax does not use the semi-colon line ending syntax, e.g. defining an enumeration (E1), a property without a default value (DProp3) or a method (Add_10). A semi-colon is needed when a property is provided with a default value (DProp).

```
EXAMPLE16
⎕cself←'cse'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'enum E1{a=10, bcd=100, efq=2}' ⍝Enumeration: No semi-colon
⎕dr⎕←⎕cse 'GetValue' 'E1.bcd'
⎕cse 'ExecStmt' 'double DProp{get;set;}=1.234;'
⎕cse 'ExecStmt' 'double DProp3{get;set;}'
```

```
⍞dr⍞←⍞cse 'GetValue' 'DProp'
⍞cse 'ExecStmt' 'double Add_10(double d){return 10+d;}' ⍝Method: No semi-colon
⍞dr⍞←⍞cse 'GetValue' 'Add_10(100)'
⍞cse 'Close'
```



C Sharp includes many key words

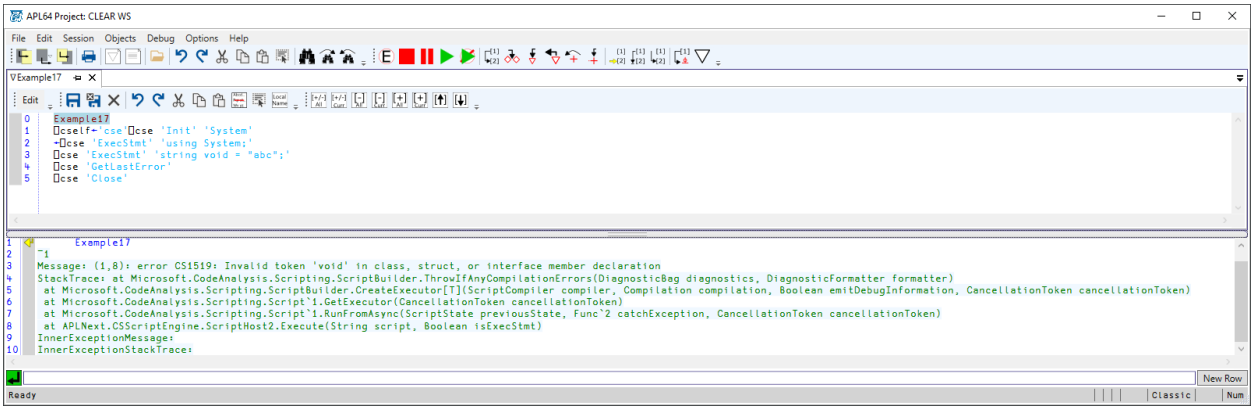
Since C# includes many key words, which are case sensitive, avoid naming C# objects with a name which is also a C# key word.

Example #17:

In C#, ‘void’ is a key word, so trying to name a C# variable ‘void’ may generate a C# debugger error.

```
Example17
⍞cself←'cse'⍞cse 'Init' 'System'
←⍞cse 'ExecStmt' 'using System;'
```

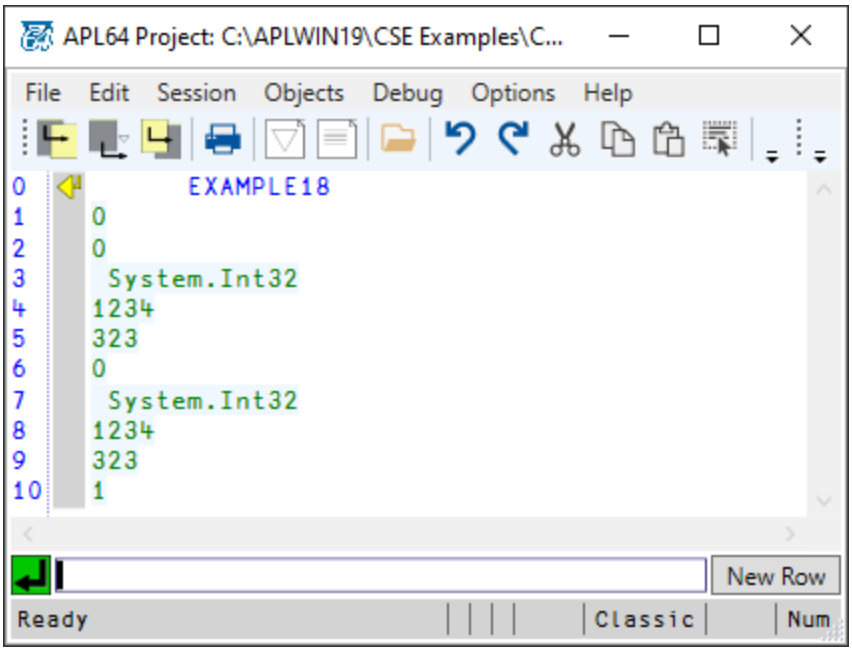
```
cse 'ExecStmt' 'string void = "abc";'
cse 'GetLastError'
cse 'Close'
```



Example #18:

In C# the key words ‘Int32’ and ‘int’ are synonymous in C#:

```
EXAMPLE18
cself←'cse'cse 'Init' 'System'
cse 'ExecStmt' 'using System;'
cse 'ExecStmt' 'Int32 I1 = 1234;'
cse 'GetObjectType' 'I1'
dr←cse 'GetValue' 'I1'
cse 'ExecStmt' 'int I2 = 1234;'
cse 'GetObjectType' 'I2'
dr←cse 'GetValue' 'I2'
cse 'GetValue' 'I1==I2'
cse 'Close'
```



String and character are distinct .Net data types

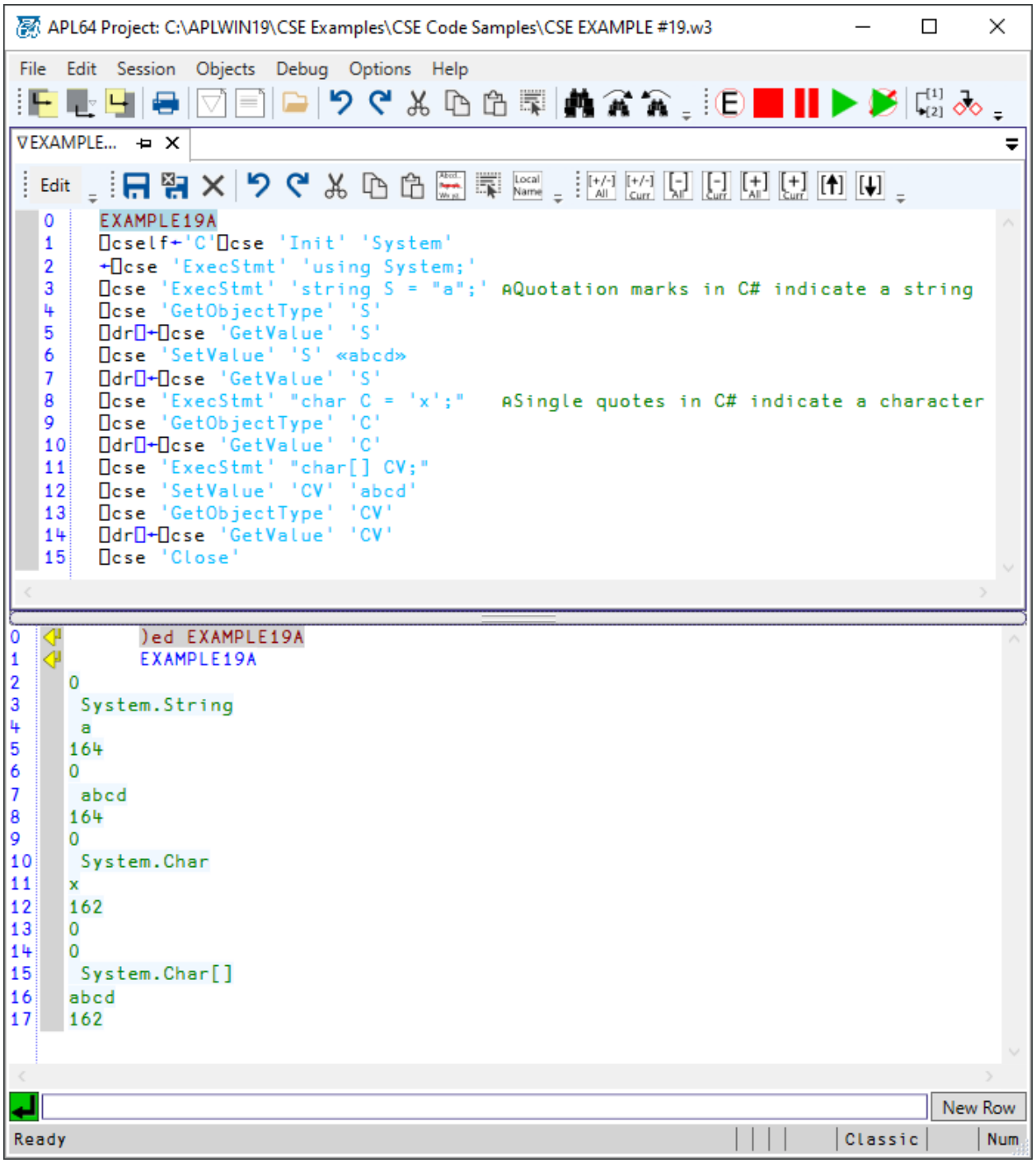
Whereas APL64 treats matched single quotes the same as matched quotation marks, in C# matched quotation marks are used to indicate string values and matched single quotes are used to indicate a character value because in .Net strings and characters are different data types. For more technical information review this [article about the string data type in C# and .Net.]

```
EXAMPLE19
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'

□cse 'ExecStmt' 'string S2 = "a";' ⍝Quotation marks in C# indicate a string
□cse 'GetObjectType' 'S2'
□dr□←□cse 'GetValue' 'S2'

□cse 'ExecStmt' "char C2 = 'x';"  ⍝Single quotes in C# indicate a character
□cse 'GetObjectType' 'C2'
□dr□←□cse 'GetValue' 'C2'

□cse 'ExecStmt' "string S1 = 'a';" ⍝Single quotes in C# indicate a character
□cse                                                              'GetLastError'
□cse 'ExecStmt' 'char C1 = "x";'  ⍝Quotation marks in C# indicate a string□cse 'GetLastError'
□cse 'Close'
```

Example #19A:

This example sets the value of C# variables of type string and character.

```
EXAMPLE19A
□cself←'C'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'string S = "a";' ⍝Quotation marks in C# indicate a string
□cse 'GetObjectType' 'S'
□dr□←□cse 'GetValue' 'S'
□cse 'SetValue' 'S' «abcd»
□dr□←□cse 'GetValue' 'S'
□cse 'ExecStmt' "char C = 'x';"   ⍝Single quotes in C# indicate a character
□cse 'GetObjectType' 'C'
□dr□←□cse 'GetValue' 'C'
□cse 'ExecStmt' "char[] CV;"
□cse 'SetValue' 'CV' 'abcd'
□cse 'GetObjectType' 'CV'
□dr□←□cse 'GetValue' 'CV'
□cse 'Close'
```

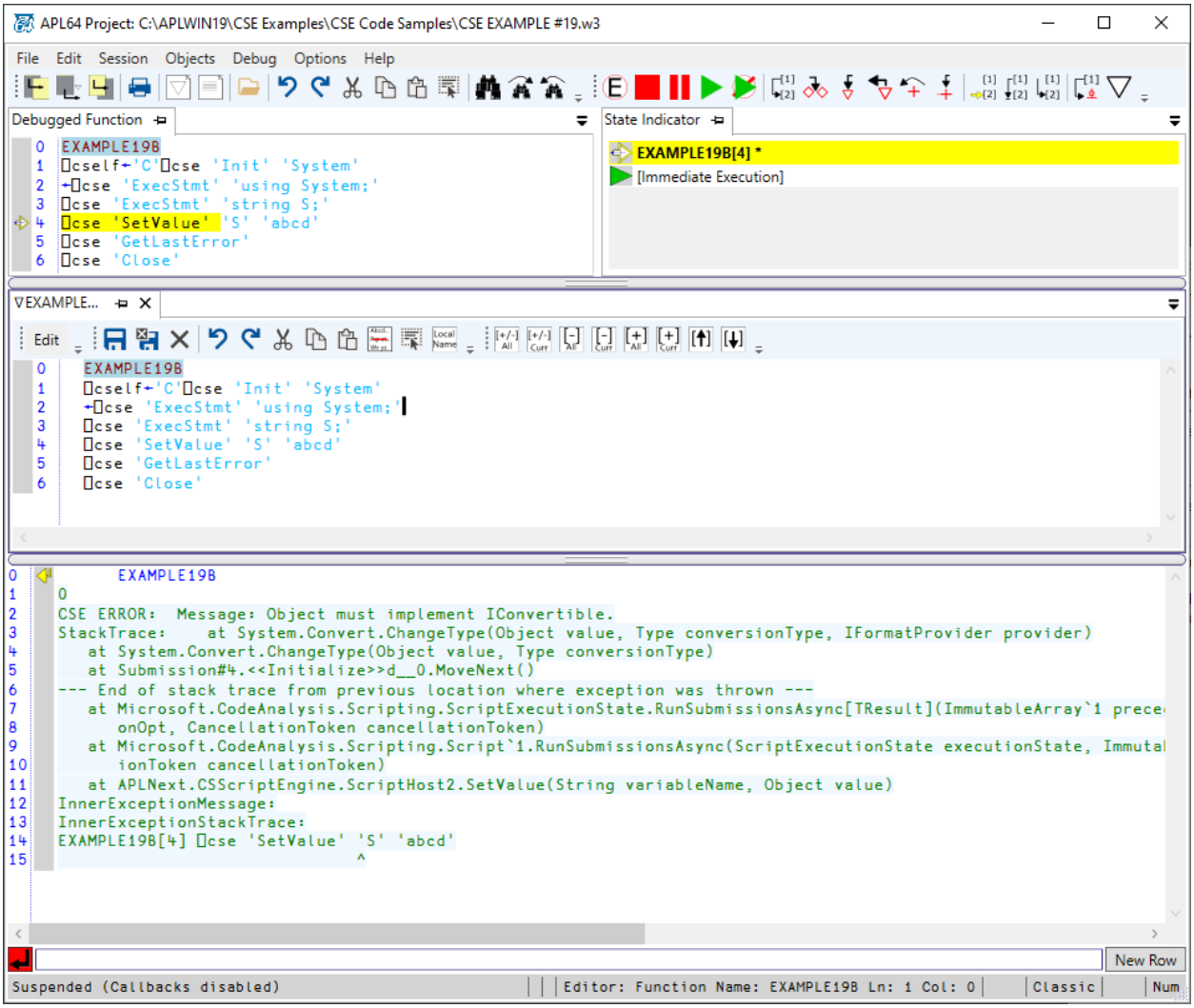


Example #19B

The value of a strongly-typed String variable cannot be set to a character vector.

```
⍞←cself←'C'⍞←cse 'Init' 'System'
←⍞←cse 'ExecStmt' 'using System;'
⍞←cse 'ExecStmt' 'string S;'
```

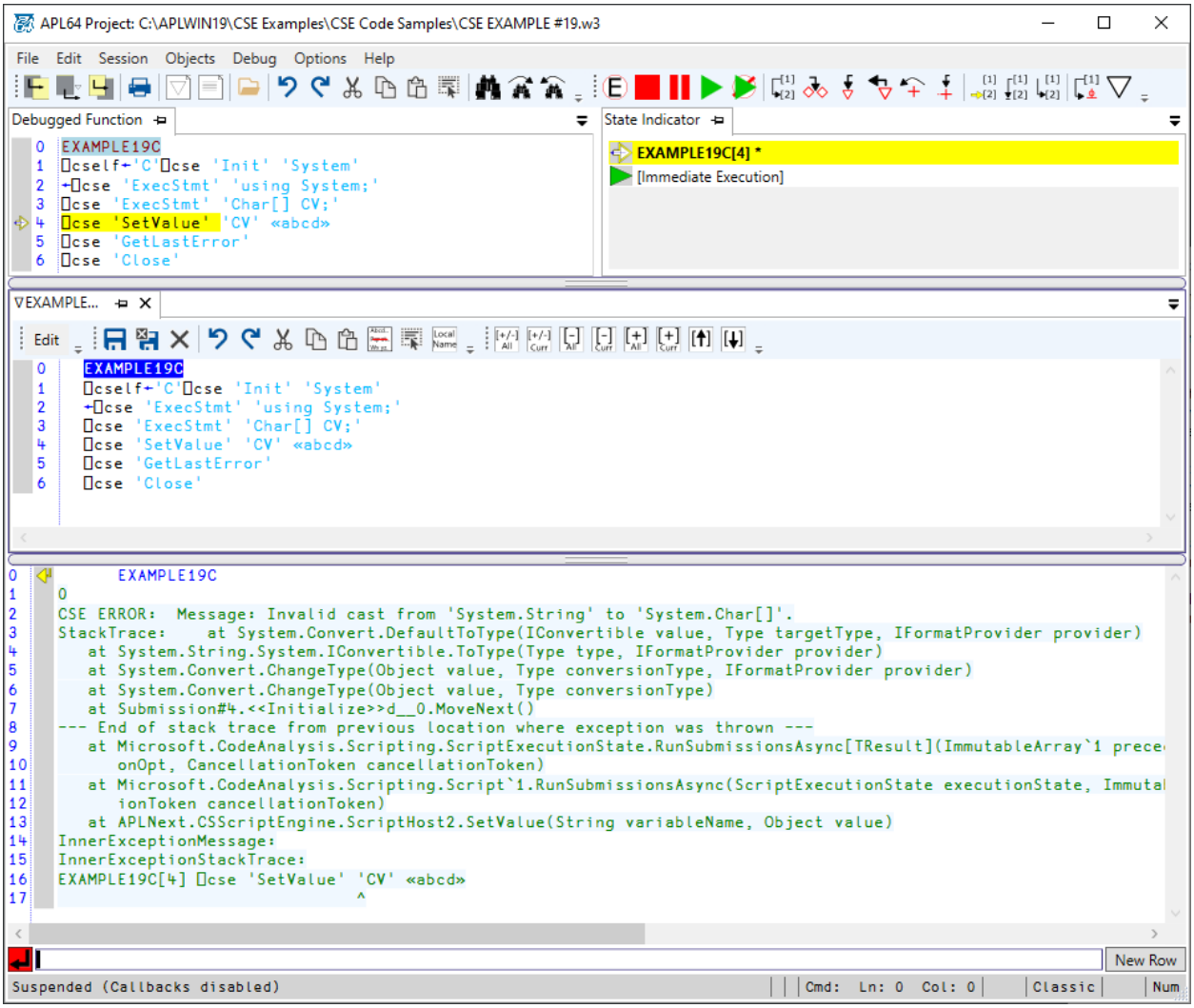
```
⎕cse 'SetValue' 'S' 'abcd'
⎕cse 'GetLastError'
⎕cse 'Close'
```



Example #19C

The value of a strongly-typed character array cannot be set to a string.

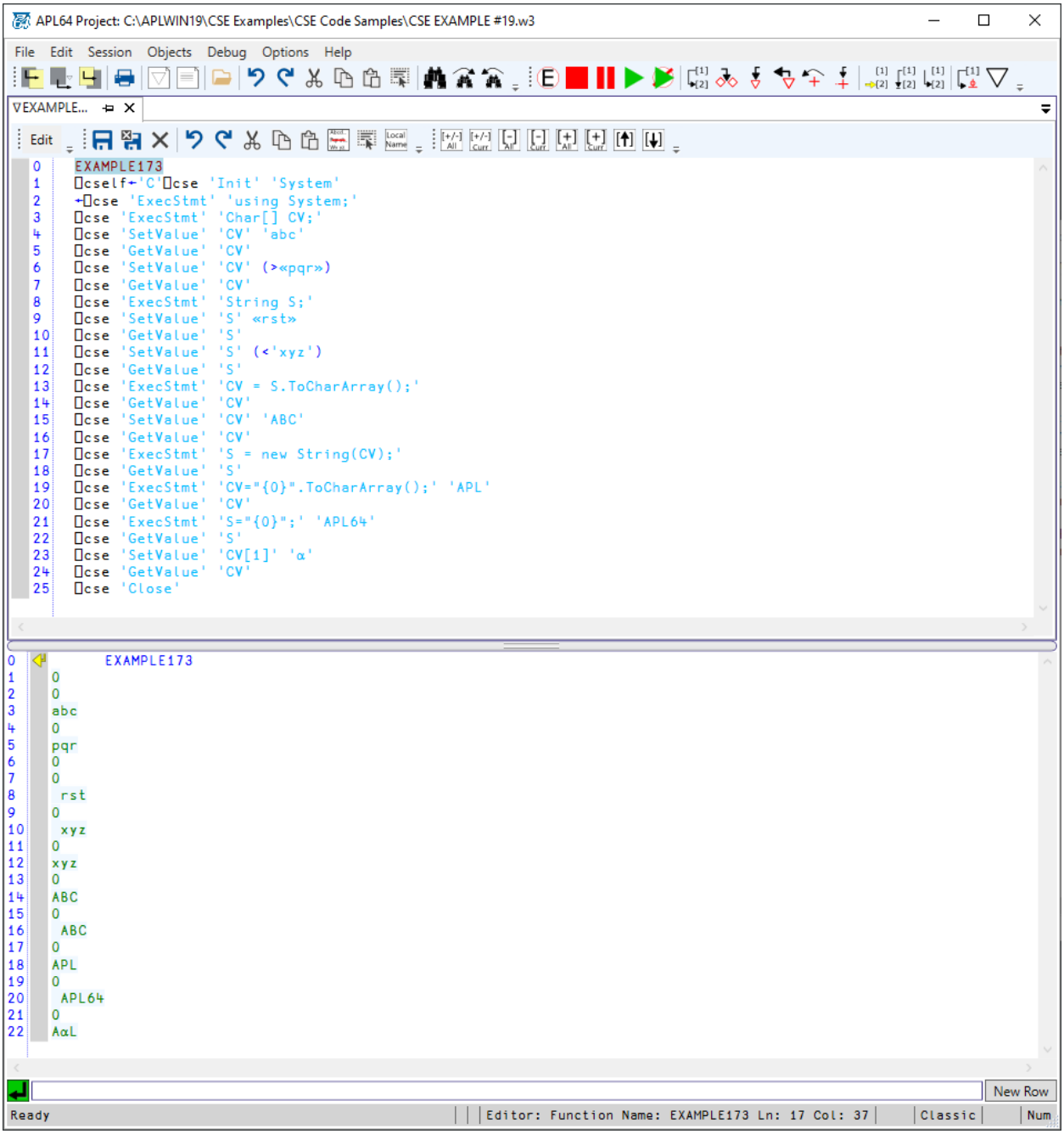
```
EXAMPLE19C
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'Char[] CV;'
⎕cse 'SetValue' 'CV' «abcd»
⎕cse 'GetLastError'
⎕cse 'Close'
```



Example #173:

This example illustrates several ways to set and get values of a C# character array and a C# string.

```
EXAMPLE173
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'Char[] CV;'
⎕cse 'SetValue' 'CV' 'abc'
⎕cse 'GetValue' 'CV'
⎕cse 'SetValue' 'CV' (>«pqr»)
⎕cse 'GetValue' 'CV'
⎕cse 'ExecStmt' 'String S;'
⎕cse 'SetValue' 'S' «rst»
⎕cse 'GetValue' 'S'
⎕cse 'SetValue' 'S' (<'xyz')
⎕cse 'GetValue' 'S'
⎕cse 'ExecStmt' 'CV = S.ToCharArray();'
⎕cse 'GetValue' 'CV'
⎕cse 'SetValue' 'CV' 'ABC'
⎕cse 'GetValue' 'CV'
⎕cse 'ExecStmt' 'S = new String(CV);'
⎕cse 'GetValue' 'S'
⎕cse 'ExecStmt' 'CV="{0}".ToCharArray();' 'APL'
⎕cse 'GetValue' 'CV'
⎕cse 'ExecStmt' 'S="{0}";' 'APL64'
⎕cse 'GetValue' 'S'
⎕cse 'SetValue' 'CV[1]' 'α'
⎕cse 'GetValue' 'CV'
⎕cse 'Close'
```



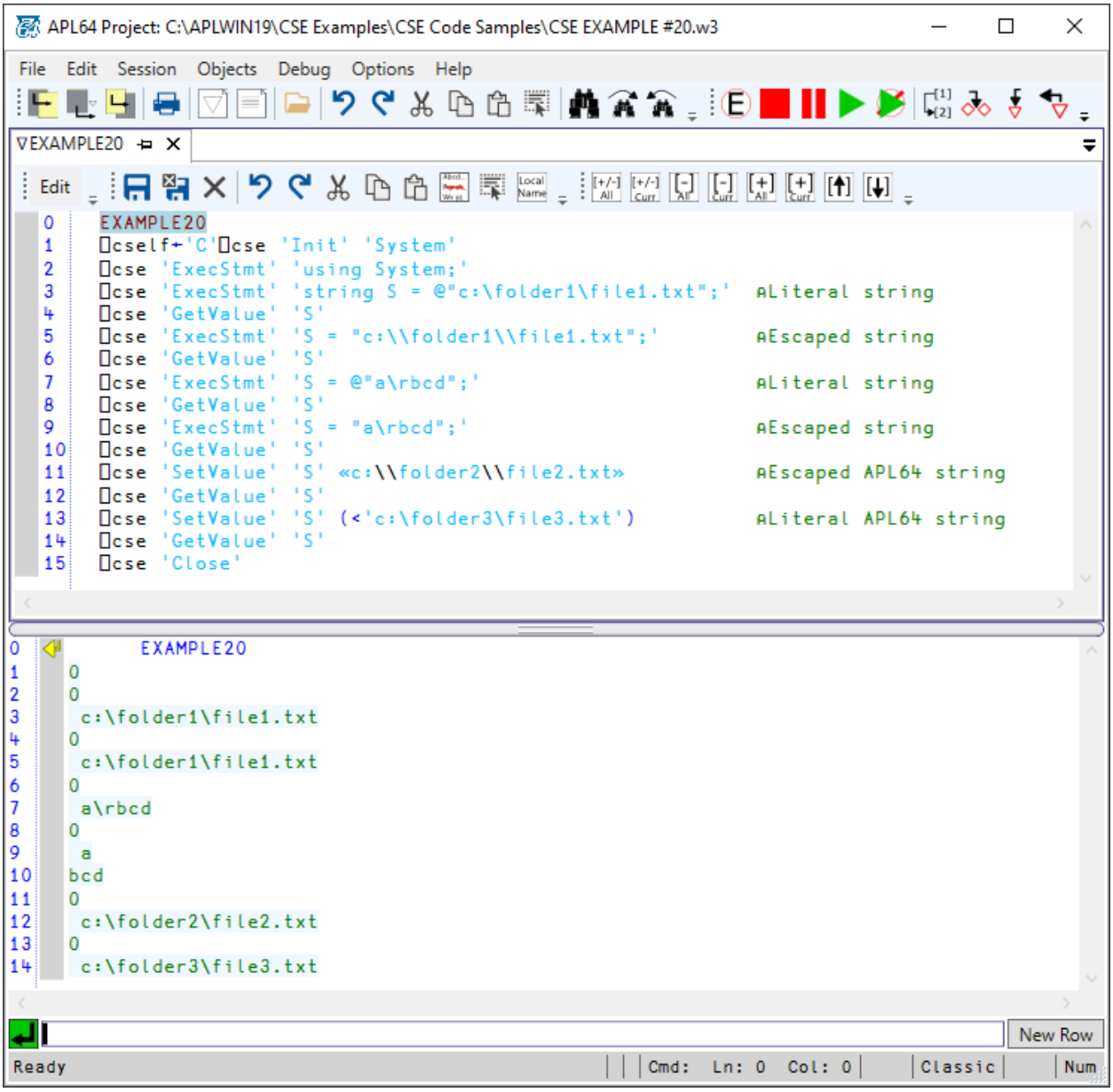
Defining Strings with embedded escape codes

C# strings can be defined as literal strings with the definition prefixed by "@" or escaped strings without the '@" prefix. A literal string is assumed to not include [C# escape codes.]

Example #20:

```
EXAMPLE20
⎕cself←'C'⎕cse 'Init' 'System'
```

```
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string S = @"c:\folder1\file1.txt";'  ⍝Literal string
⎕cse 'GetValue' 'S'
⎕cse 'ExecStmt' 'S = "c:\\folder1\\file1.txt";'        ⍝Escaped string
⎕cse 'GetValue' 'S'
⎕cse 'ExecStmt' 'S = @"a\rbcd";'                       ⍝Literal string
⎕cse 'GetValue' 'S'
⎕cse 'ExecStmt' 'S = "a\rbcd";'                        ⍝Escaped string
⎕cse 'GetValue' 'S'
⎕cse 'SetValue' 'S' «c:\\folder2\\file2.txt»           ⍝Escaped APL64 string
⎕cse 'GetValue' 'S'
⎕cse 'SetValue' 'S' (<'c:\folder3\file3.txt')          ⍝Literal APL64 string
⎕cse 'GetValue' 'S'
⎕cse 'Close'
```



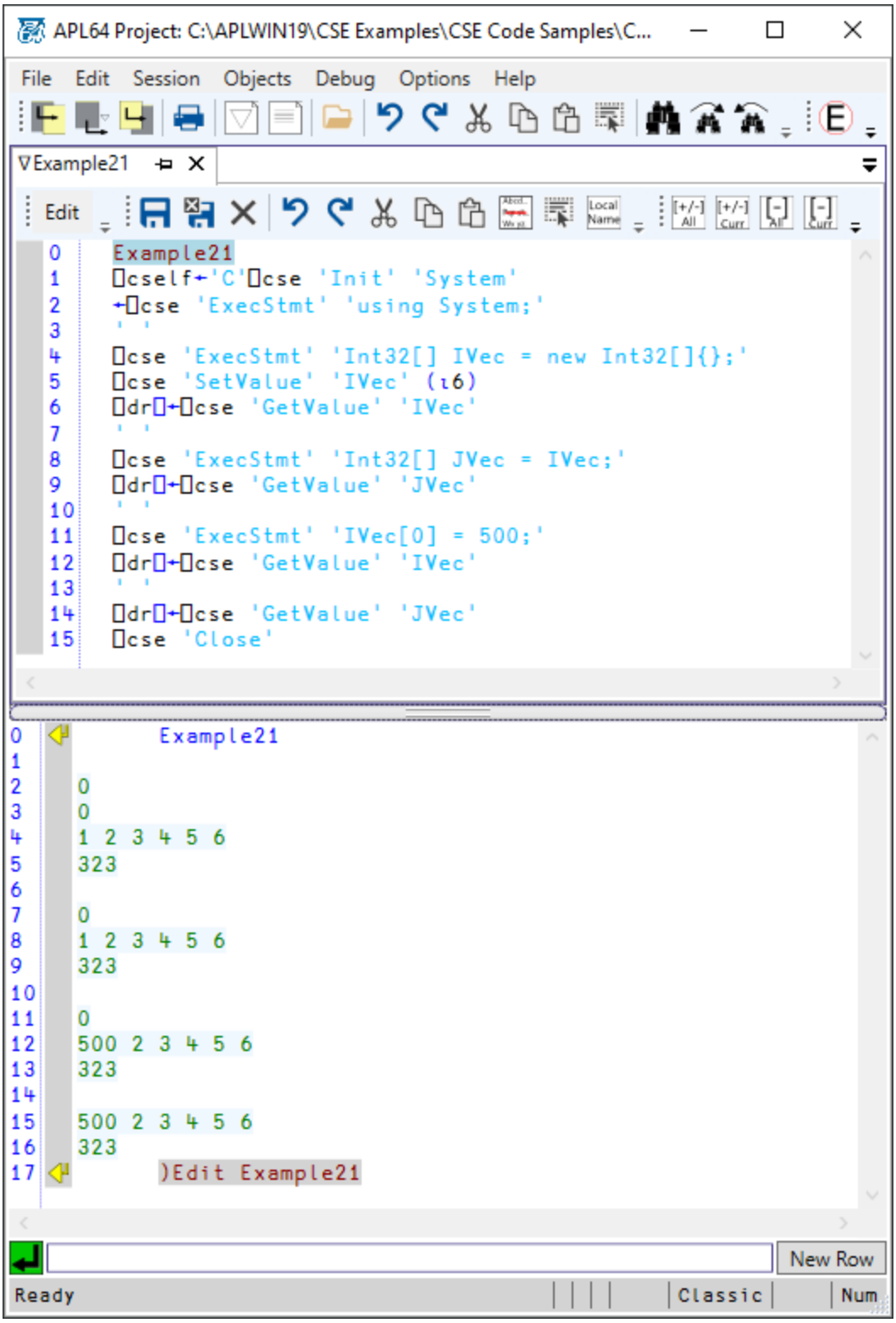
C Sharp Uses Assignment by Reference

Since the C# assignment operator ("=") operates using references, a new C# variable assigned to be a previously-defined C# variable, will be a reference to that previously-defined C# variable. Modifications made to the data of the previously-defined variable will affect the new C# variable. This behavior does not apply to .Net scalar, value types, such as Int32, double, bool, string, and char.

Example #21:

In this example the C# Int32 vector 'IVec' is created and the C# Int32 vector 'JVec' is assigned by reference to be 'IVec'. When a data element of 'IVec' is modified, since 'JVec' is a reference to 'IVec' the analogous data element in 'JVec' is also changed.

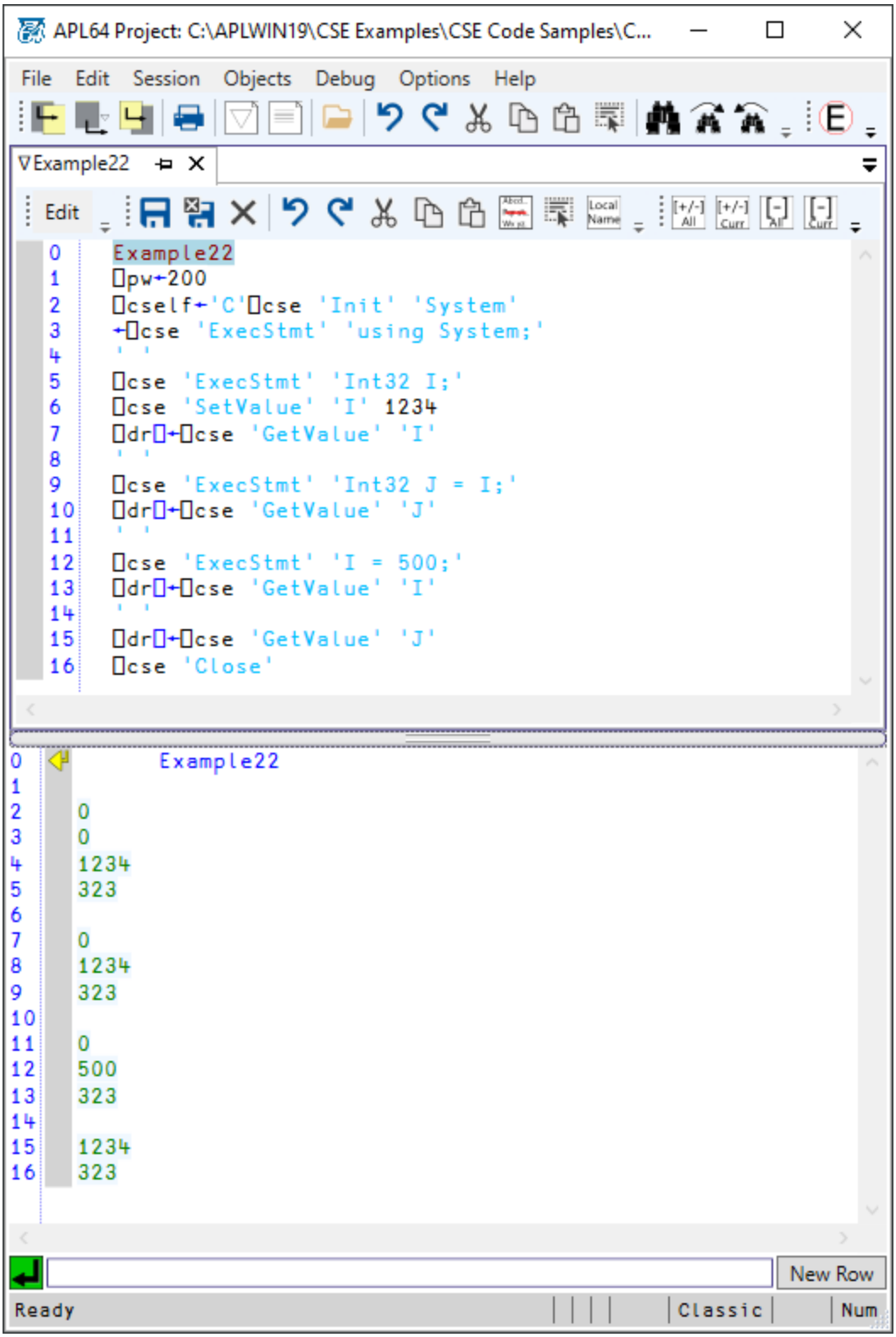
```
Example21
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
' '
⎕cse 'ExecStmt' 'Int32[] IVec = new Int32[]{};'
⎕cse 'SetValue' 'IVec' (⍳6)
⎕dr⎕←⎕cse 'GetValue' 'IVec'
' '
⎕cse 'ExecStmt' 'Int32[] JVec = IVec;'
⎕dr⎕←⎕cse 'GetValue' 'JVec'
' '
⎕cse 'ExecStmt' 'IVec[0] = 500;'
dr⎕←⎕cse 'GetValue' 'IVec'
' '
⎕dr⎕←⎕cse 'GetValue' 'JVec'
⎕cse 'Close'
```



Example #22:

In this example .Net scalar, value Int32 types are illustrated. Although the C# variable 'J' is assigned to the C# variable 'I', when 'I' is modified 'J' is not because they are scalar integers.

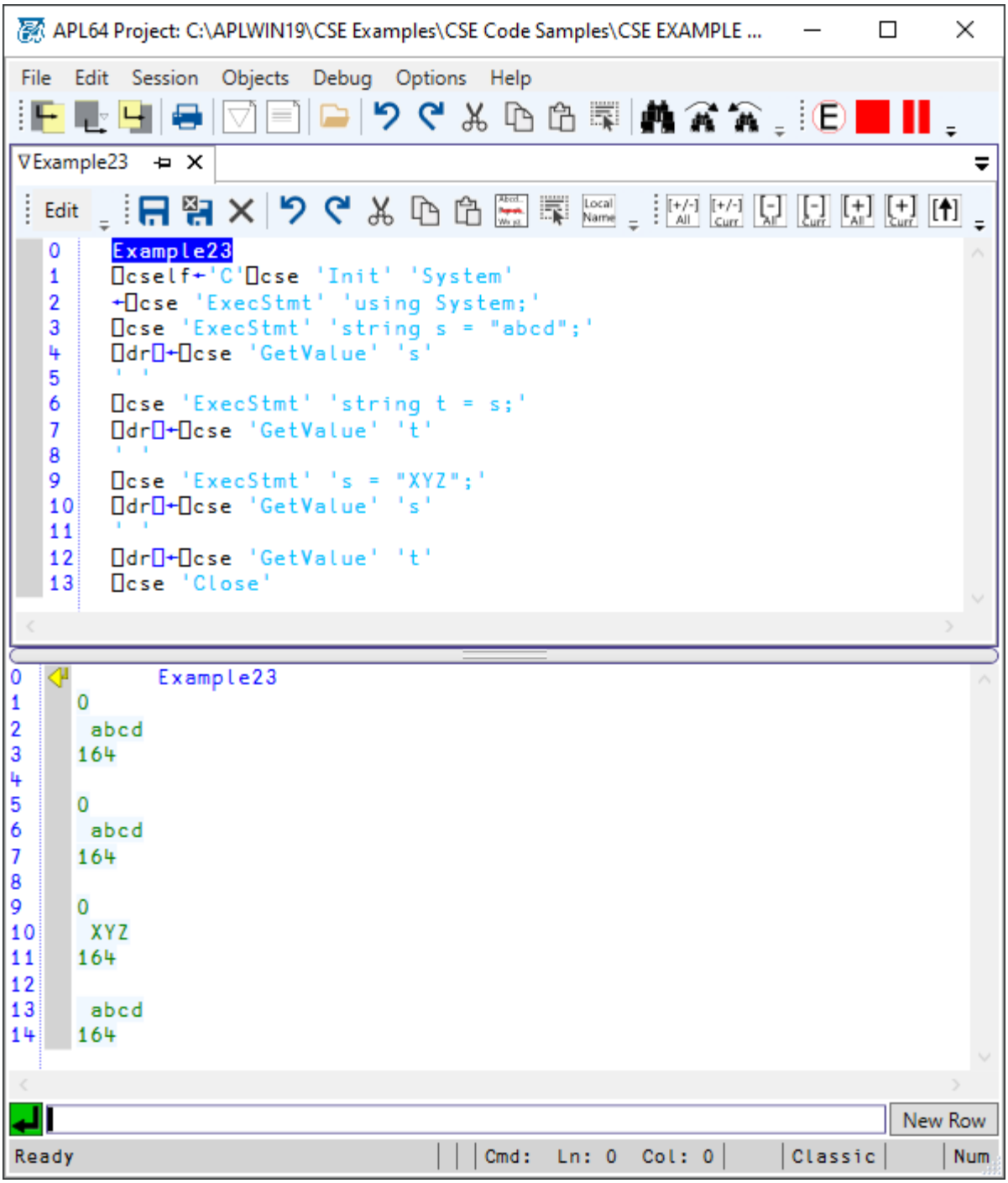
```
Example22
⎕pw←200
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
' '
⎕cse 'ExecStmt' 'Int32 I;'
⎕cse 'SetValue' 'I' 1234
⎕dr⎕←⎕cse 'GetValue' 'I'
' '
⎕cse 'ExecStmt' 'Int32 J = I;'
⎕dr⎕←⎕cse 'GetValue' 'J'
' '
⎕cse 'ExecStmt' 'I = 500;'⎕dr⎕←⎕cse 'GetValue' 'I'
' '
⎕dr⎕←⎕cse 'GetValue' 'J'
⎕cse 'Close'
```


APL64 Project: C:\APLWIN19\CSE Examples\CSE Code Samples\C...
File Edit Session Objects Debug Options Help
Example22
Edit
0 Example22
1 ⎕pw←200
2 ⎕cself←'C'⎕cse 'Init' 'System'
3 ←⎕cse 'ExecStmt' 'using System;'
4 ' '
5 ⎕cse 'ExecStmt' 'Int32 I;'
6 ⎕cse 'SetValue' 'I' 1234
7 ⎕dr⎕←⎕cse 'GetValue' 'I'
8 ' '
9 ⎕cse 'ExecStmt' 'Int32 J = I;'
10 ⎕dr⎕←⎕cse 'GetValue' 'J'
11 ' '
12 ⎕cse 'ExecStmt' 'I = 500;'
13 ⎕dr⎕←⎕cse 'GetValue' 'I'
14 ' '
15 ⎕dr⎕←⎕cse 'GetValue' 'J'
16 ⎕cse 'Close'
0 Example22
1
2 0
3 0
4 1234
5 323
6
7 0
8 1234
9 323
10
11 0
12 500
13 323
14
15 1234
16 323
New Row
Ready
Classic
Num

Example #23:

In this example .Net scalar string variables are used to illustrate that assignment by reference does not apply to them.

```
Example23
⍝cself←'C'⍝cse 'Init' 'System'
←⍝cse 'ExecStmt' 'using System;'
⍝cse 'ExecStmt' 'string s = "abcd";'
⍝dr⍝←⍝cse 'GetValue' 's'
' '
⍝cse 'ExecStmt' 'string t = s;'
⍝dr⍝←⍝cse 'GetValue' 't'
' '
⍝cse 'ExecStmt' 's = "XYZ";'
⍝dr⍝←⍝cse 'GetValue' 's'
' '
⍝dr⍝←⍝cse 'GetValue' 't'
⍝cse 'Close'
```

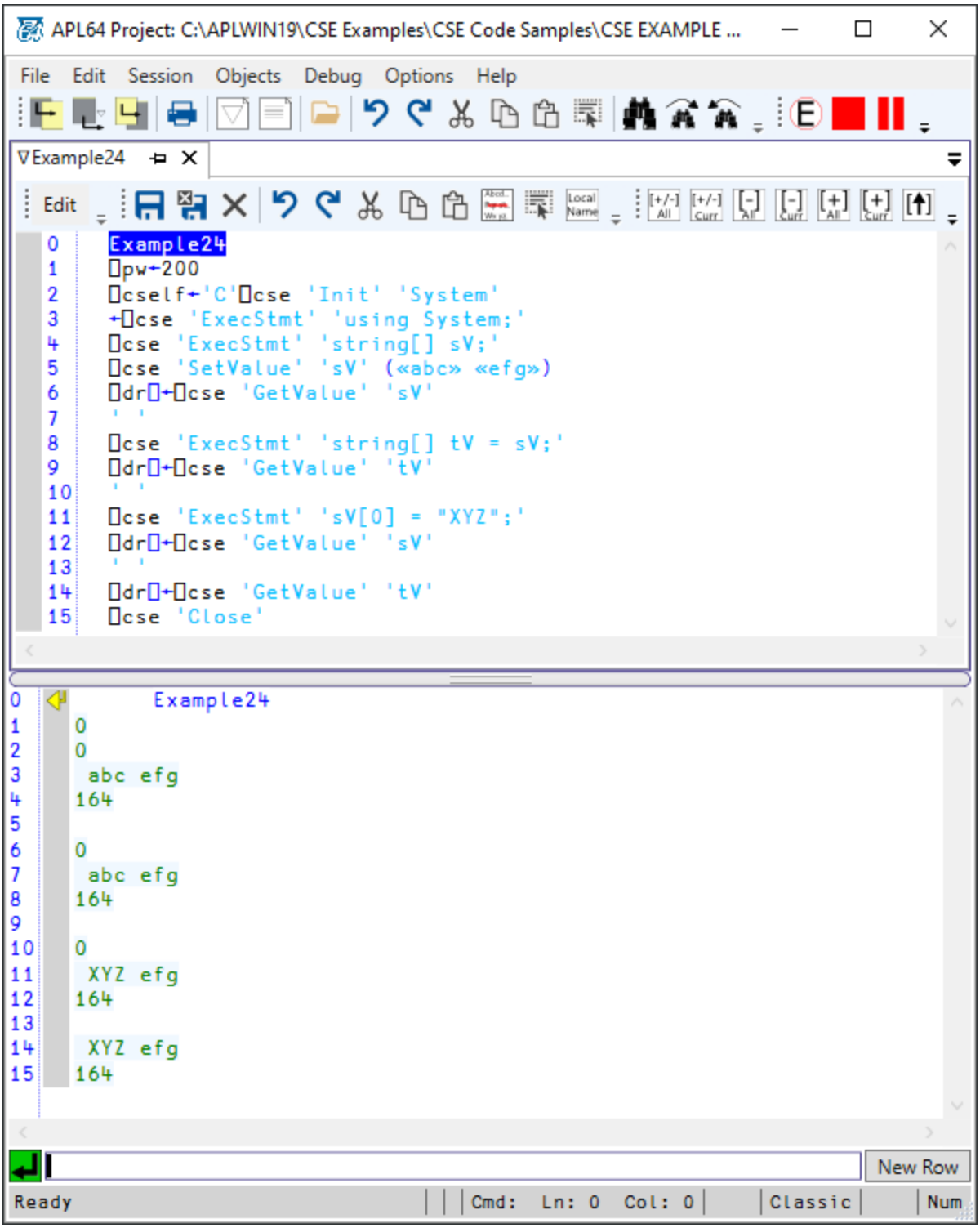


Example #24:

In this example vectors of strings are used which are not scalars in C#, so assignment by reference applies to them.

```
Example24
⎕pw←200
```

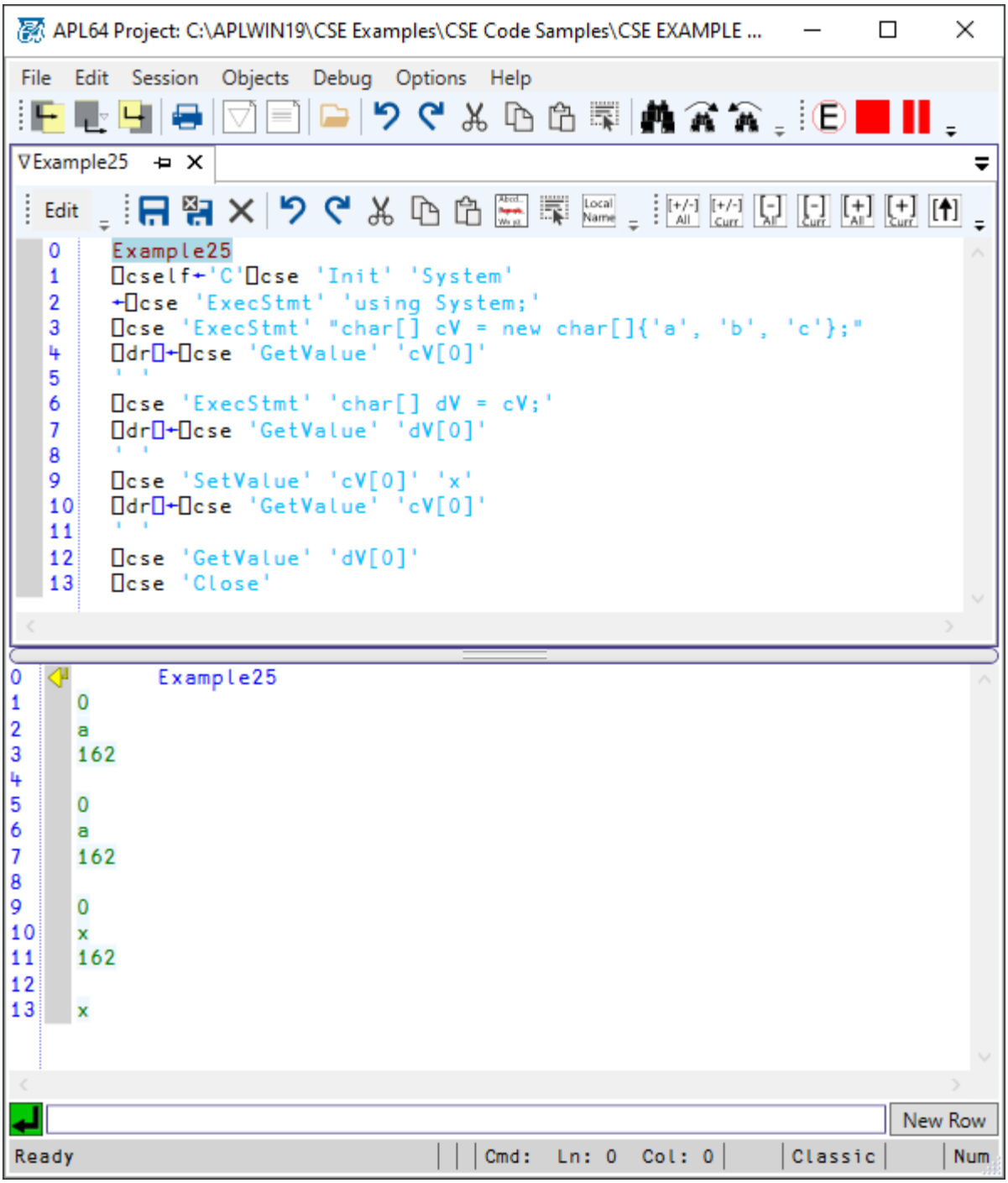
```
⎕cself←'C'⎕cse                                  'Init'                  'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string[] sV;'
⎕cse 'SetValue' 'sV' («abc» «efg»)
⎕dr⎕←⎕cse 'GetValue' 'sV'
' '
⎕cse 'ExecStmt' 'string[] tV = sV;'
⎕dr⎕←⎕cse 'GetValue' 'tV'
' '
⎕cse 'ExecStmt' 'sV[0] = "XYZ";'
⎕dr⎕←⎕cse 'GetValue' 'sV'
' '
⎕dr⎕←⎕cse 'GetValue' 'tV'
⎕cse 'Close'
```



Example #25:

In this example .Net character vectors are used, which are not scalars in C#, so assignment by reference applies to them.

```
Example25
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' "char[] cV = new char[]{'a', 'b', 'c'};"
⎕dr⎕←⎕cse 'GetValue' 'cV[0]'
' '
⎕cse 'ExecStmt' 'char[] dV = cV;'
⎕dr⎕←⎕cse 'GetValue' 'dV[0]'
' '
⎕cse 'SetValue' 'cV[0]' 'x'
⎕dr⎕←⎕cse 'GetValue' 'cV[0]'
' '
⎕cse 'GetValue' 'dV[0]'
⎕cse 'Close'
```



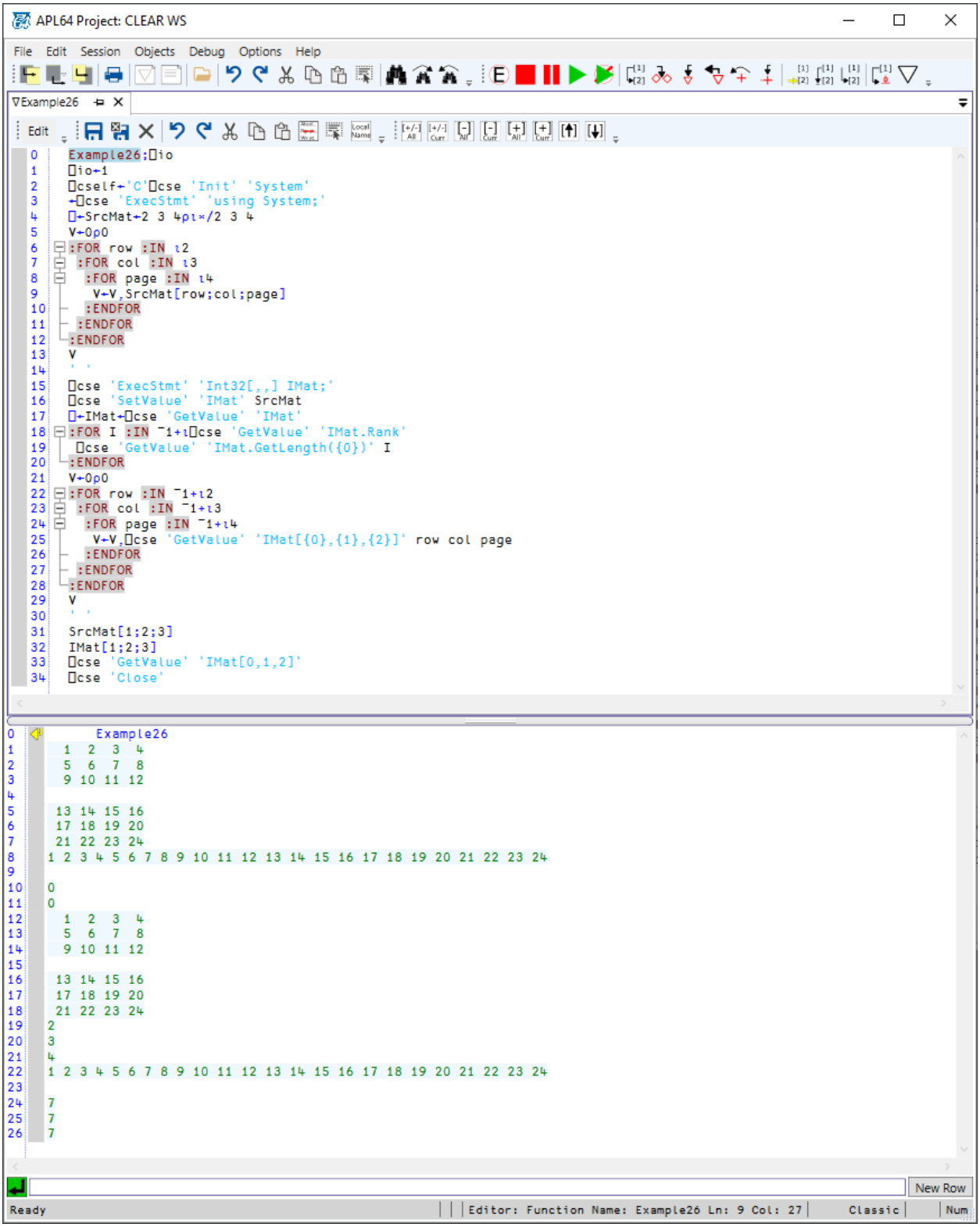
C Sharp Row and Column Order are different from that in APL64

The row and column order of a C# array are different from that in APL64 and the index origin for specifying a row or a column of a C# array is zero. The APL64 CSE automatically handles the row and column order difference when SetValue or GetValue is used to set or get an entire array. The APL64 programmer must provide the applicable element indices and index origin when manipulating elements of APL64 or C# array.

Example #26:

This example illustrates these behaviors.

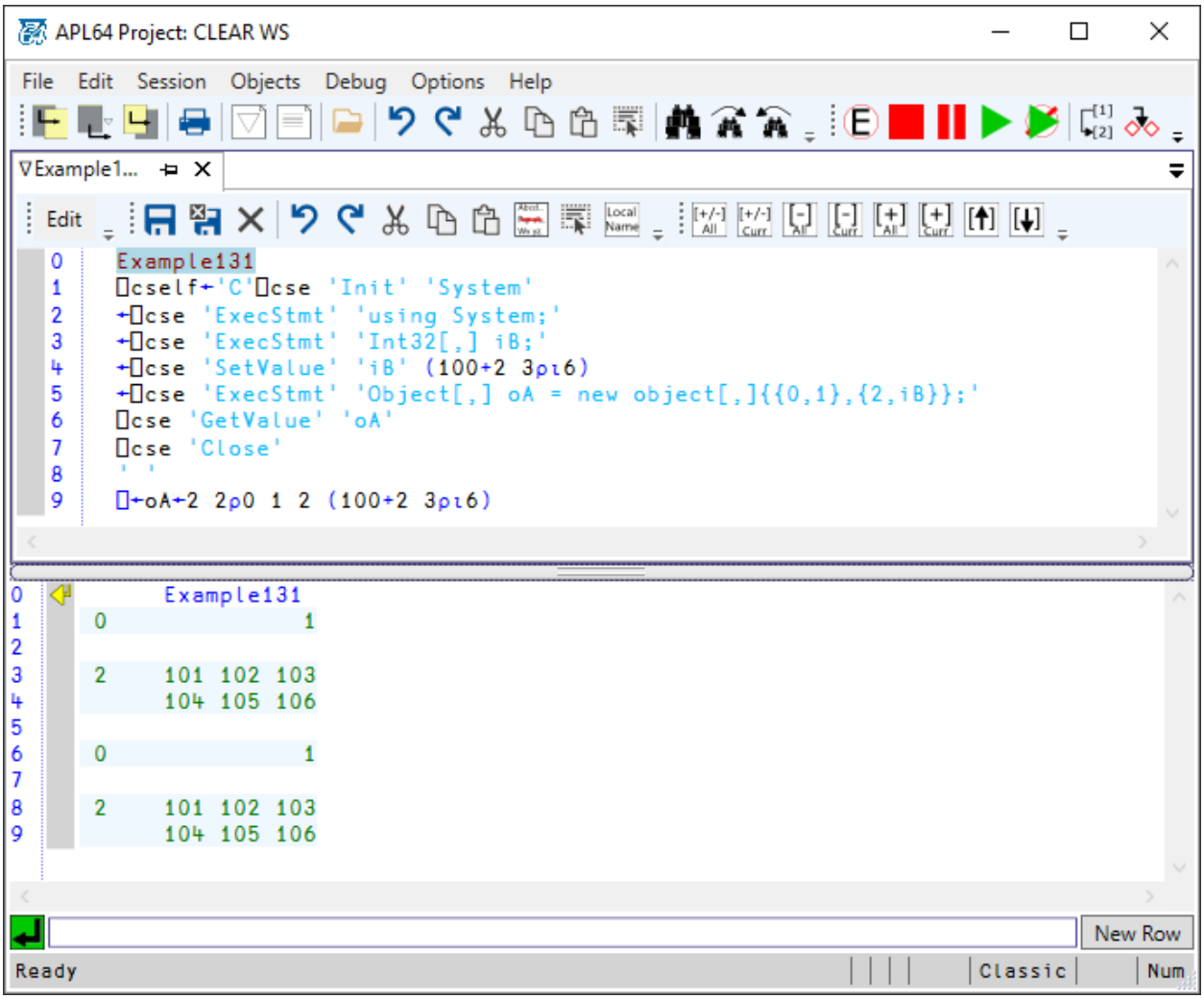
```
Example26;⎕io
⎕io←1
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕←SrcMat←2 3 4⍴⍳×/2 3 4
V←0⍴0
:FOR row :IN ⍳2
 :FOR col :IN ⍳3
  :FOR page :IN ⍳4
   V←V,SrcMat[row;col;page]
  :ENDFOR
 :ENDFOR
:ENDFOR
V
' '
⎕cse 'ExecStmt' 'Int32[,,] IMat;'
⎕cse 'SetValue' 'IMat' SrcMat
⎕←IMat←⎕cse 'GetValue' 'IMat'
:FOR I :IN ¯1+⍳⎕cse 'GetValue' 'IMat.Rank'
 ⎕cse 'GetValue' 'IMat.GetLength({0})' I
:ENDFOR
V←0⍴0
:FOR row :IN ¯1+⍳2
 :FOR col :IN ¯1+⍳3
  :FOR page :IN ¯1+⍳4
   V←V,⎕cse 'GetValue' 'IMat[{0},{1},{2}]' row col page
  :ENDFOR
 :ENDFOR
:ENDFOR
V
' '
SrcMat[1;2;3]
IMat[1;2;3]
⎕cse 'GetValue' 'IMat[0,1,2]'
⎕cse 'Close'
```



Example #131

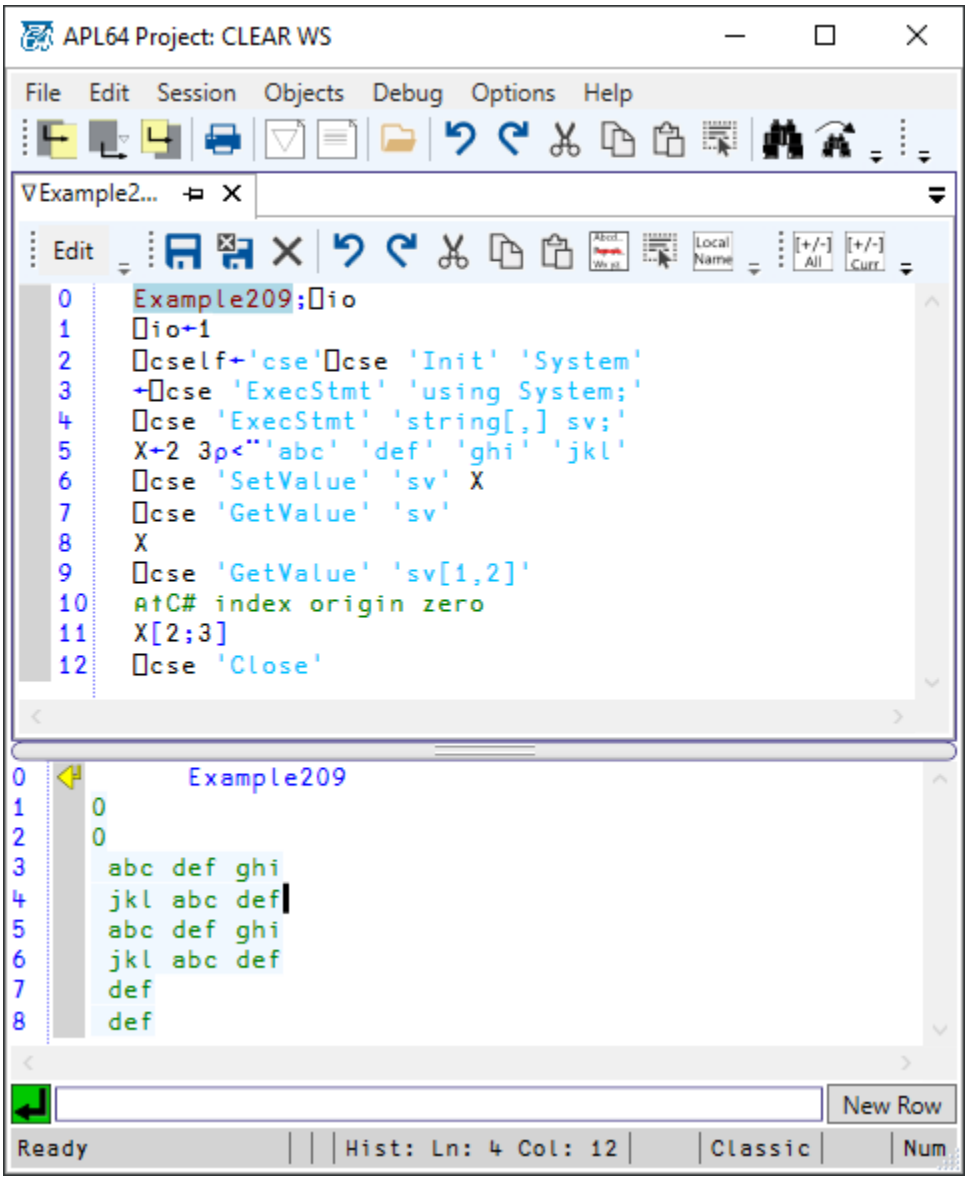
This example illustrates the automatic handling by the APL64 CSE of the row and column order of a nested array.

```
Example131
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
←⎕cse 'ExecStmt' 'Int32[,] iB;'
←⎕cse 'SetValue' 'iB' (100+2 3⍴⍳6)
←⎕cse 'ExecStmt' 'Object[,] oA = new object[,]{{0,1},{2,iB}};'
⎕cse 'GetValue' 'oA'
⎕cse 'Close'
' '
⎕←oA←2 2⍴0 1 2 (100+2 3⍴⍳6)
```



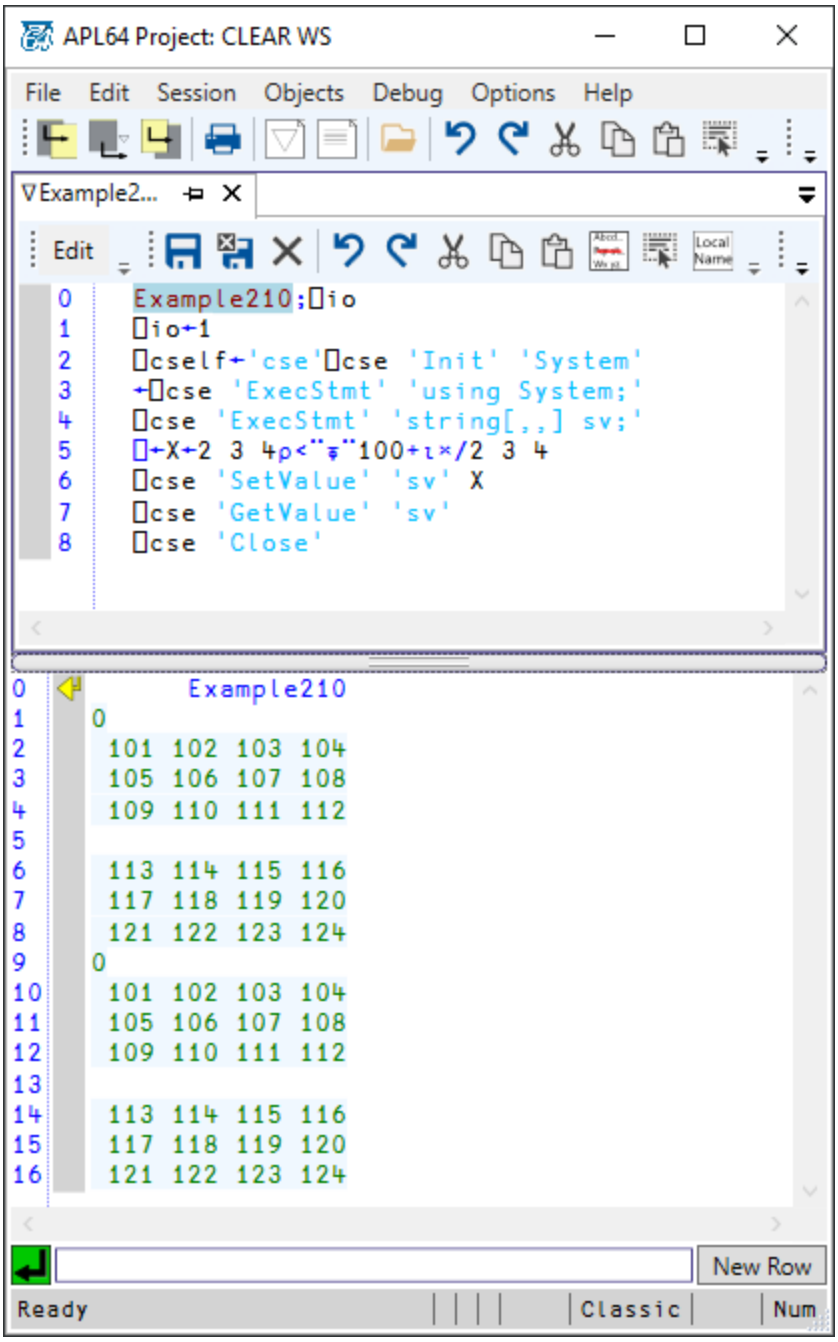
Example #209:
C# arrays of strings

```
Example209;⎕io
⎕io←1
⎕cself←'cse'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string[,] sv;'
X←2 3⍴⍳'abc' 'def' 'ghi' 'jkl'
⎕cse 'SetValue' 'sv' X
⎕cse 'GetValue' 'sv'
X
⎕cse 'GetValue' 'sv[1,2]'
⍝↑C# index origin zero
X[2;3]
⎕cse 'Close'
```



Example #210:
Rank 3 string array

```
Example210;□io
□io←1
□cself←'cse'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'string[,,] sv;'
□←X←2 3 4⍴<¨⊤⍜¨100+⍳×/2 3 4
□cse 'SetValue' 'sv' X
□cse 'GetValue' 'sv'
□cse 'Close
```



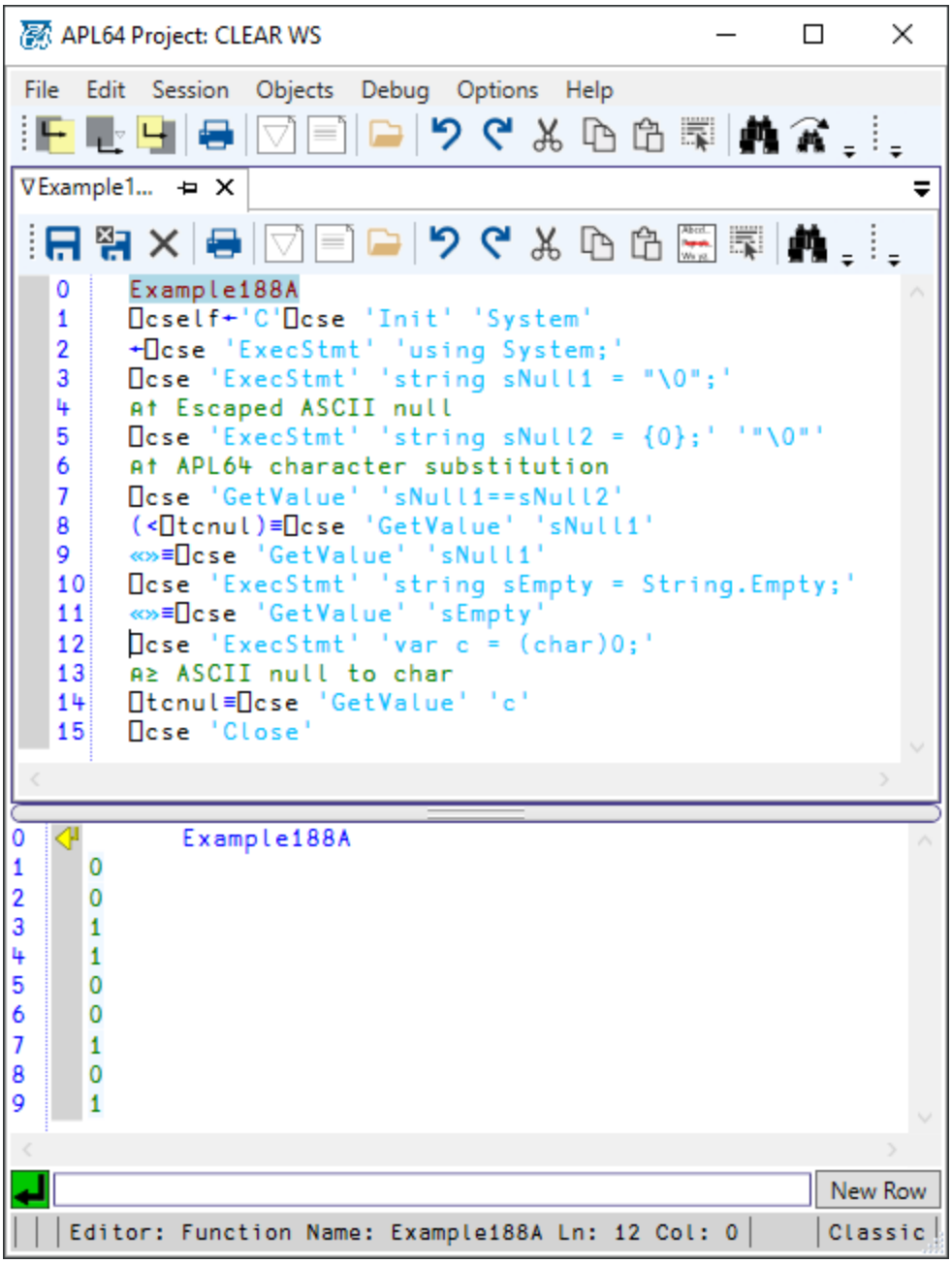
Null in .Net

In .Net 'null' is a key word. In .Net there are nullable reference types and nullable value types. The ASCII null character be (part of) the value of a .Net object.

Example #188:

Working with a .Net string variable and the ASCII null character.

```
Example188A
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string sNull1 = "\0";'
⍝↑ Escaped ASCII null
⎕cse 'ExecStmt' 'string sNull2 = {0};' '"\0"'
⍝↑ APL64 character substitution
⎕cse 'GetValue' 'sNull1==sNull2'
(<⎕tcnul)≡⎕cse 'GetValue' 'sNull1'
«»≡⎕cse 'GetValue' 'sNull1'
⎕cse 'ExecStmt' 'string sEmpty = String.Empty;'
«»≡⎕cse 'GetValue' 'sEmpty'
⎕cse 'ExecStmt' 'var c = (char)0;'
⍝≥ ASCII null to char
⎕tcnul≡⎕cse 'GetValue' 'c'
⎕cse 'Close'
```

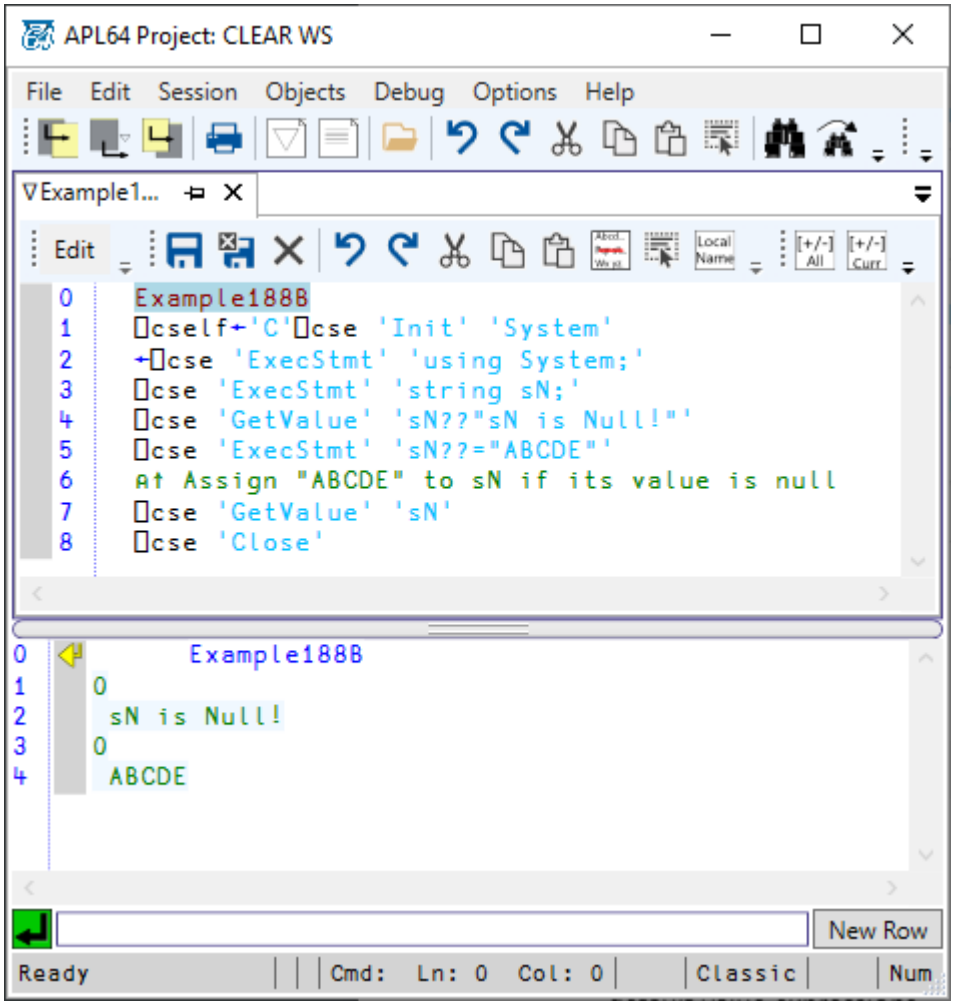


Example #188B

Checking for a null value and assigning a non-null value if the existing value is null.

```
Example188B
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string sN;'
⎕cse 'GetValue' 'sN??"sN is Null!"'
⎕cse 'ExecStmt' 'sN??="ABCDE"'
```

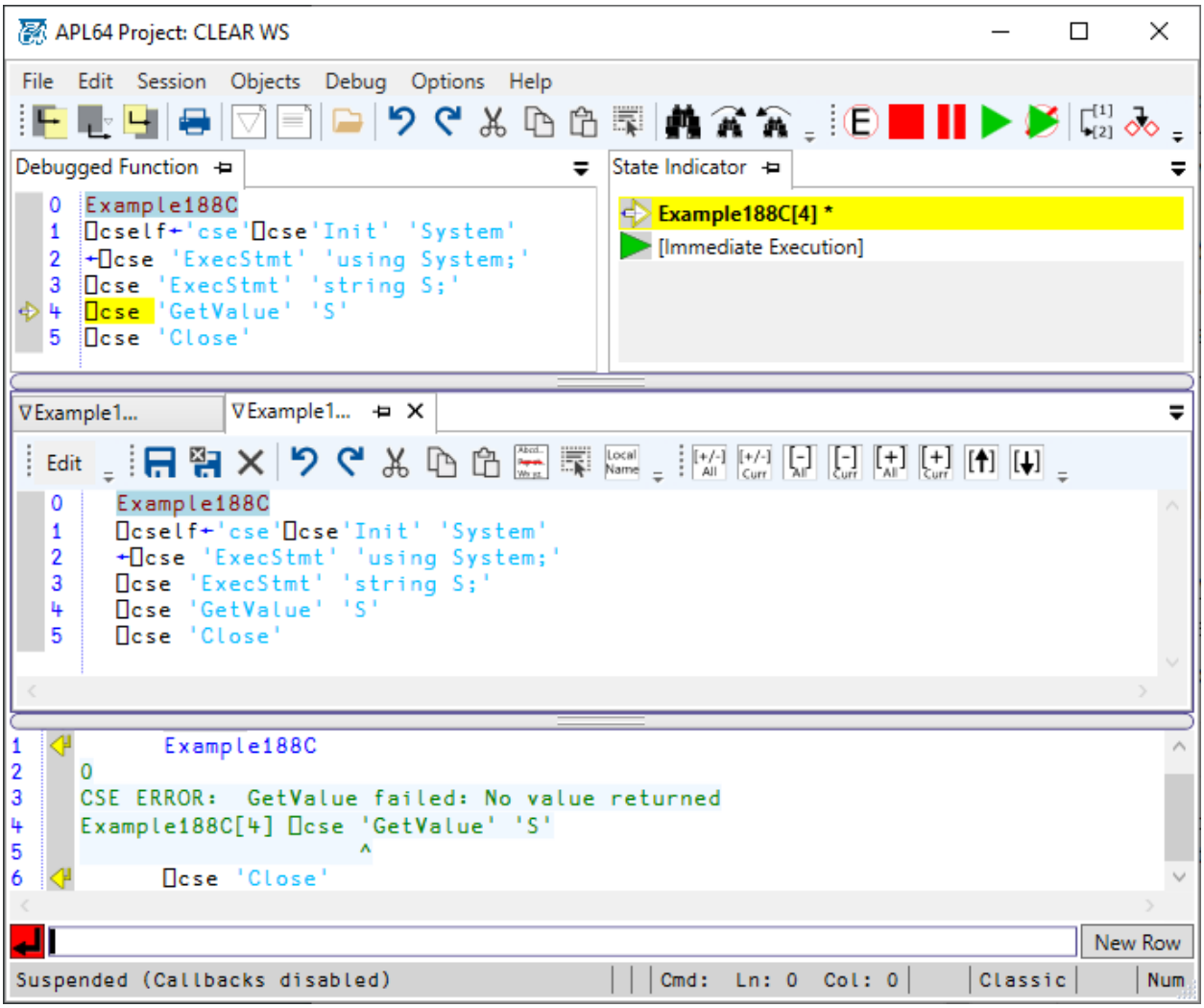
```
⍝↑ Assign "ABCDE" to sN if its value is null
□cse 'GetValue' 'sN'
□cse 'Close'
```



Example #188C

Non-nullable types need to have an assigned value before their value can be obtained by the CSE 'GetValue' instance method.

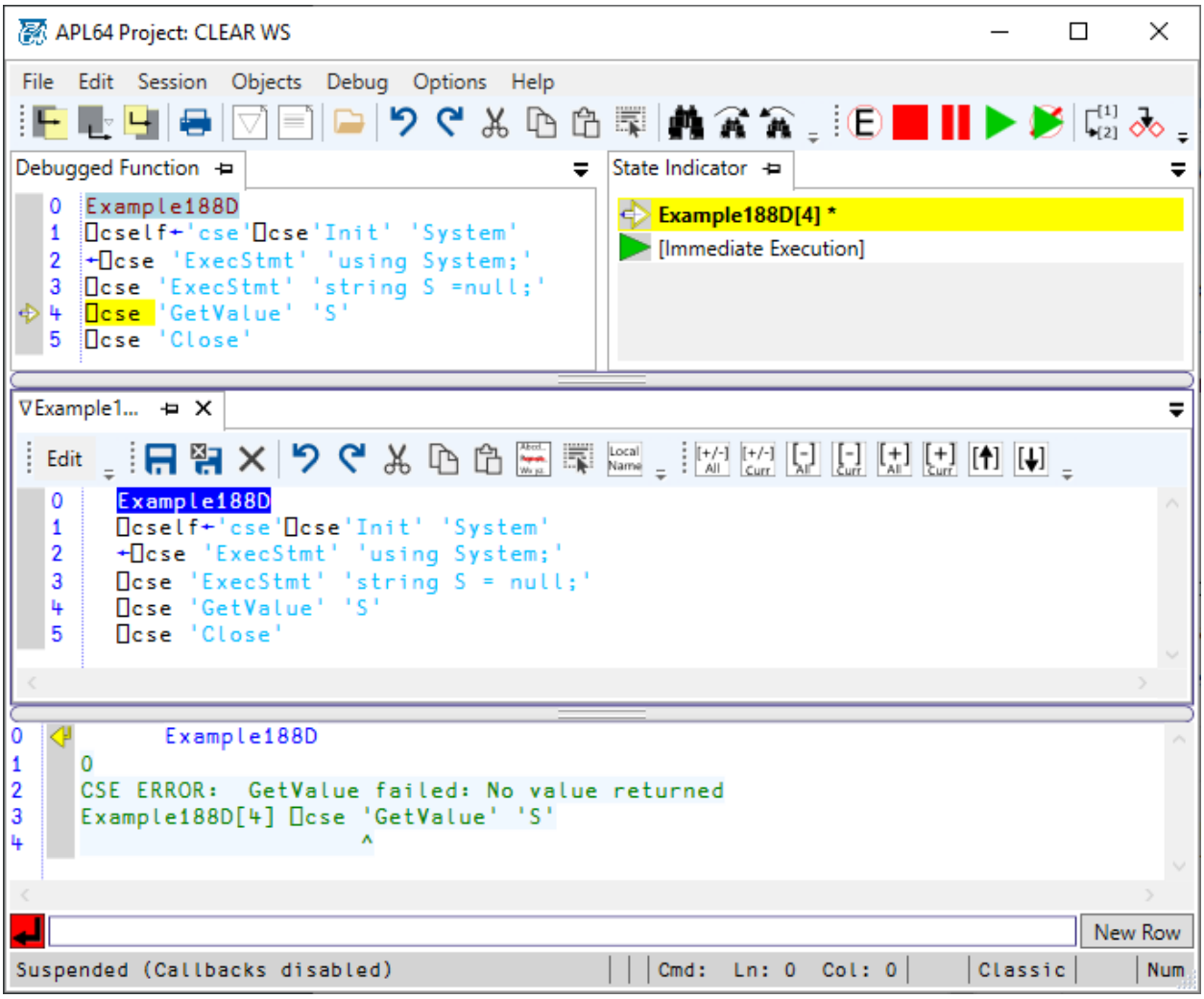
```
Example188C
□cself←'cse'□cse'Init' 'System'
←□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'string S;'
□cse 'GetValue' 'S'
□cse 'Close'
```



Example #188D

The CSE ‘GetValue’ method cannot obtain the value of a .Net variable assigned a null value, until that variable is assigned a non-null value.

```
Example188D
⎕cself←'cse'⎕cse'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string S = null;'
⎕cse 'GetValue' 'S'
⎕cse 'Close'
```



CARG System Variable

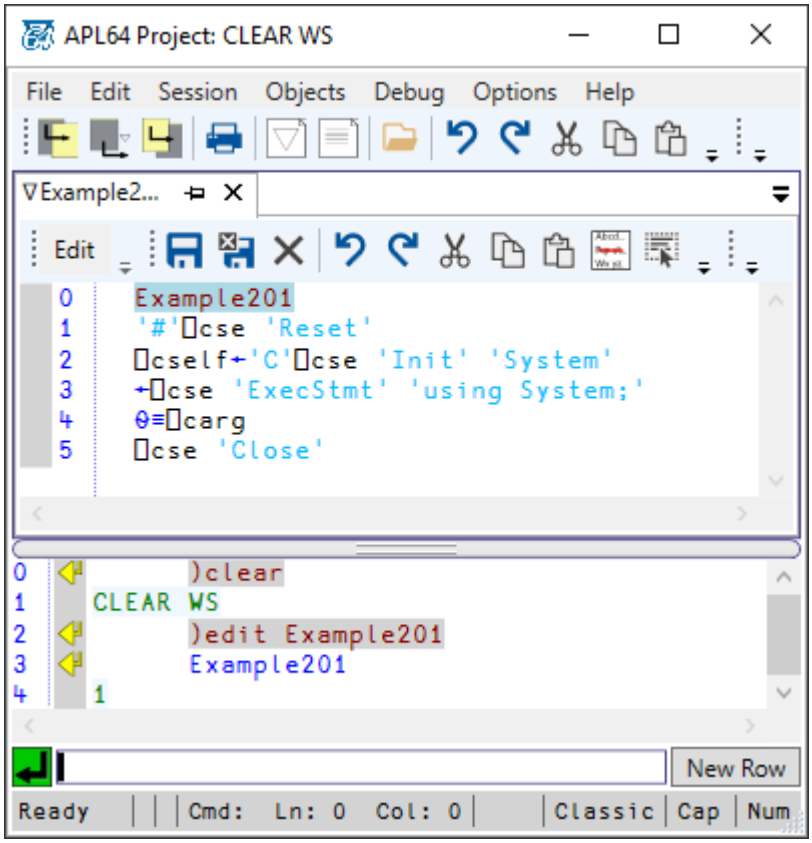
The ⎕carg system variable is used only in association with the CSE 'AddCustomEventHandler' and 'AddCustomEventHandlerEx' methods. It is a read-only system variable.

While a .Net custom event is being handled by an APL64 event handler function subscribed to that event using the CSE 'AddCustomEventHandler' and 'AddCustomEventHandlerEx' methods, that event handler function may access ⎕carg to obtain .Net event's values, set by the CSE as defined by the APL64 programmer's .Net event delegate. The value of ⎕cse is localized to the scope of the APL64 function handling the .Net custom event. Refer to the documentation of the CSE 'AddCustomEventHandler' method for examples illustrating ⎕carg.

Example #201

In a 'clear' APL64 workspace or if no .Net custom event is being handled by an APL64 event handler function subscribed to that event using the CSE 'AddCustomEventHandler' and 'AddCustomEventHandlerEx' methods, the value of ⎕carg is ⍬.

```
Example201
'#'⎕cse 'Reset'
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⍬≡⎕carg
⎕cse 'Close'
```



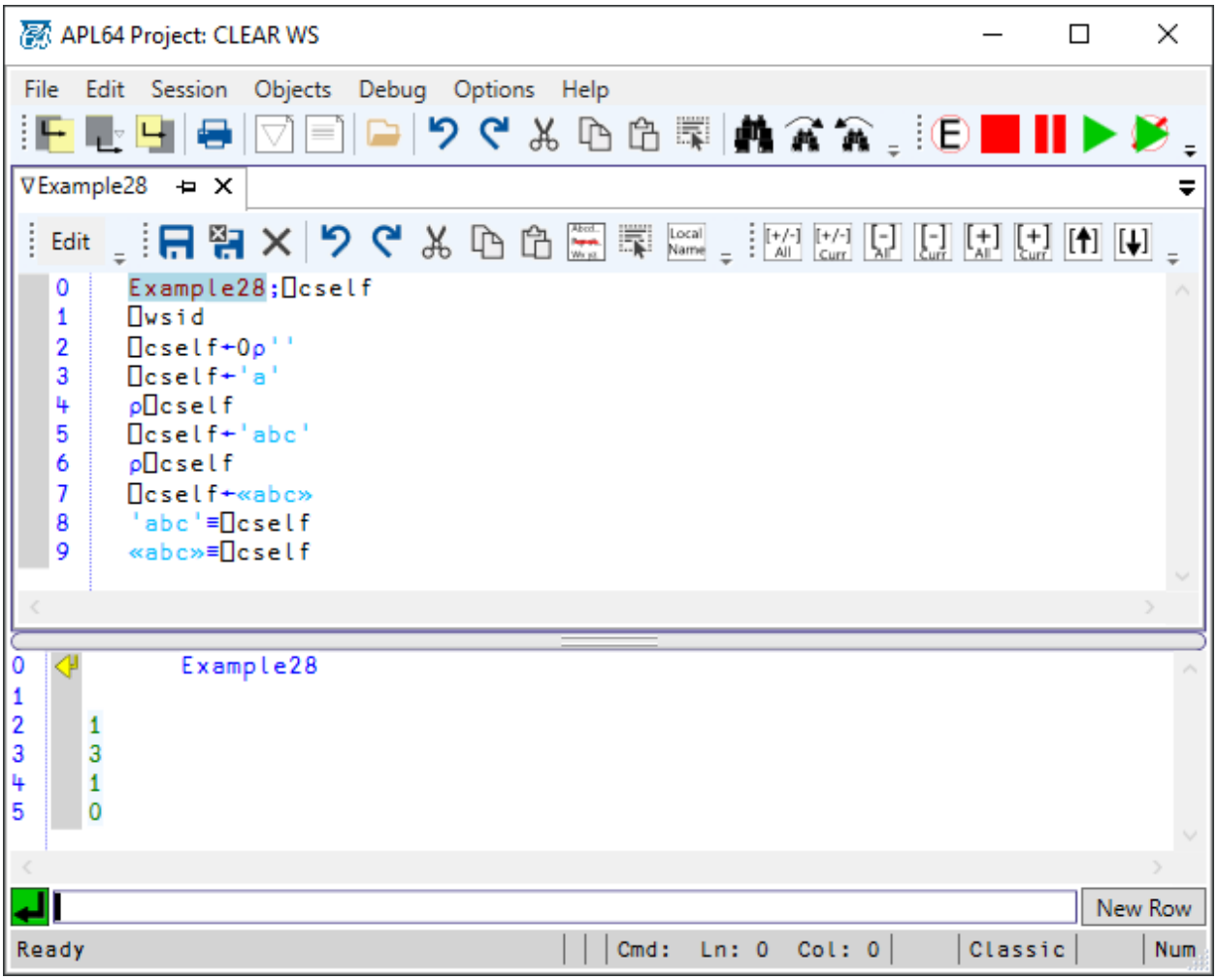
CSELF System Variable

The ⎕cself system variable, with an appropriate value, can be used in lieu of the left argument of the ⎕cse system function. The ⎕cself system variable may be localized in the header of an APL64 function. The value of ⎕cself may be assigned a character scalar, character vector or scalar string. When accessed the value of ⎕cself is always a character vector. In a clear workspace, the value of ⎕cself is 0ρ''.

Example #28:

```
Example28;⎕cself
⎕wsid
⎕cself←0ρ''
⎕cself←'a'
ρ⎕cself
⎕cself←'abc'
```

```
ρ⎕cself
⎕cself←«abc»
'abc'≡⎕cself
«abc»≡⎕cself
```



CSE System Object

The CSE System Object supports methods and properties which apply to all instances of the CSE which exist in an APL64 session. To access the CSE system object, the left argument of the □cse system function is '#'.

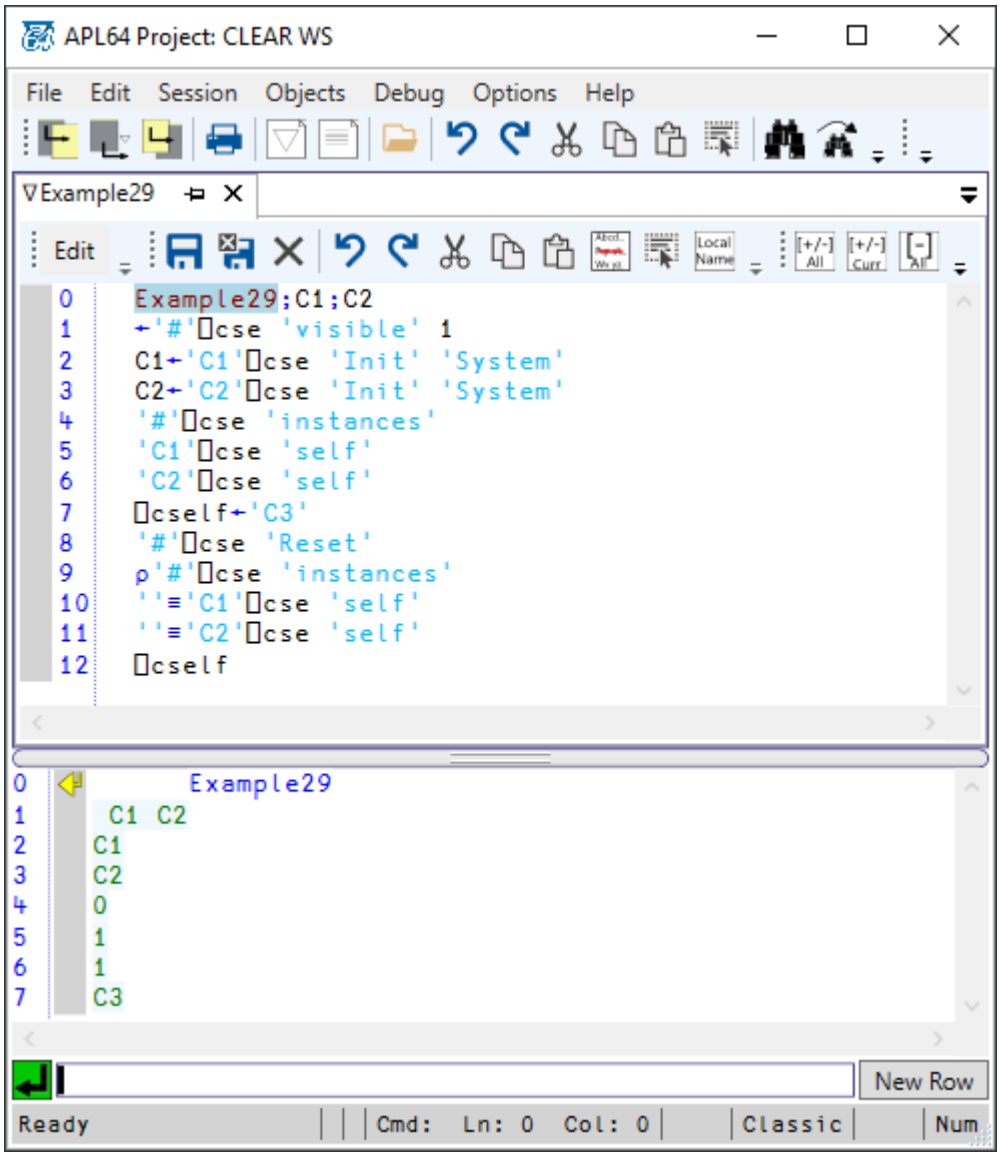
CSE System Object Methods

CSE System Object Reset method

The CSE object 'Reset' method deletes all existing instances of the CSE in the current APL64 session including the associated CSE engines. The CSE object 'Reset' method does not modify the value of the ⎕cself variable.

Example #29:

```
Example29;C1;C2
C1←'C1'⎕cse 'Init' 'System'
C2←'C2'⎕cse 'Init' 'System'
'#'⎕cse 'instances'
'C1'⎕cse 'self'
'C2'⎕cse 'self'
⎕cself←'C3'
'#'⎕cse 'Reset'
⍴'#'⎕cse 'instances'
''≡'C1'⎕cse 'self'
''≡'C2'⎕cse 'self'
⎕cself
```

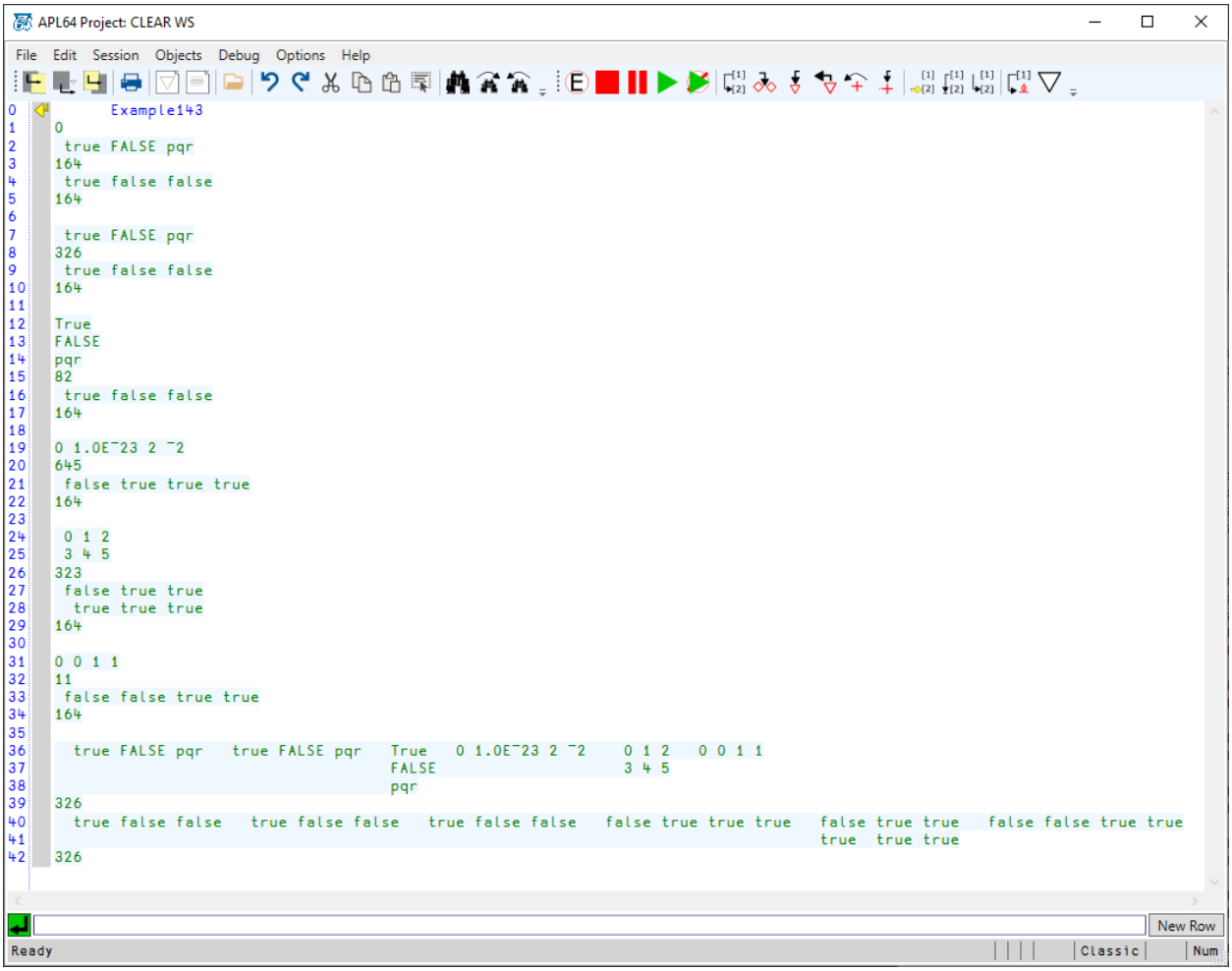


CSE System Object ToBool Method

This utility method will convert an APL64 object to an APL64 string type object suitable for use when the CSE SetValue or ExecStmt instance methods are used to set the value of a Boolean element of a .Net object.

Example #143

```
Example143
⍝cself←'C'⍝cse 'Init' 'System'
⍝cse 'ExecStmt' 'using System;'
⍝DR ⍝←X1←«true» «FALSE» «pqr»
⍝DR ⍝←'#'⍝cse 'ToBool' X1
' '
⍝DR ⍝←X2←'true' 'FALSE' 'pqr'
⍝DR ⍝←'#'⍝cse 'ToBool' X2
' '
⍝DR ⍝←X3←⍝OVERV 'True' 'FALSE' 'pqr'\
⍝DR ⍝←'#'⍝cse 'ToBool' X3
' '
⍝DR ⍝←X4←0 1E¯23 2 ¯2
⍝DR ⍝←'#'⍝cse 'ToBool' X4
' '
⍝DR ⍝←X5←2 3⍴¯1+⍳6
⍝DR ⍝←'#'⍝cse 'ToBool' X5
' '
⍝DR ⍝←X6←0 0 1 1
⍝DR ⍝←'#'⍝cse 'ToBool' X6
' '
⍝DR ⍝←X←X1 X2 X3 X4 X5 X6
⍝DR ⍝←'#'⍝cse 'ToBool' X
```



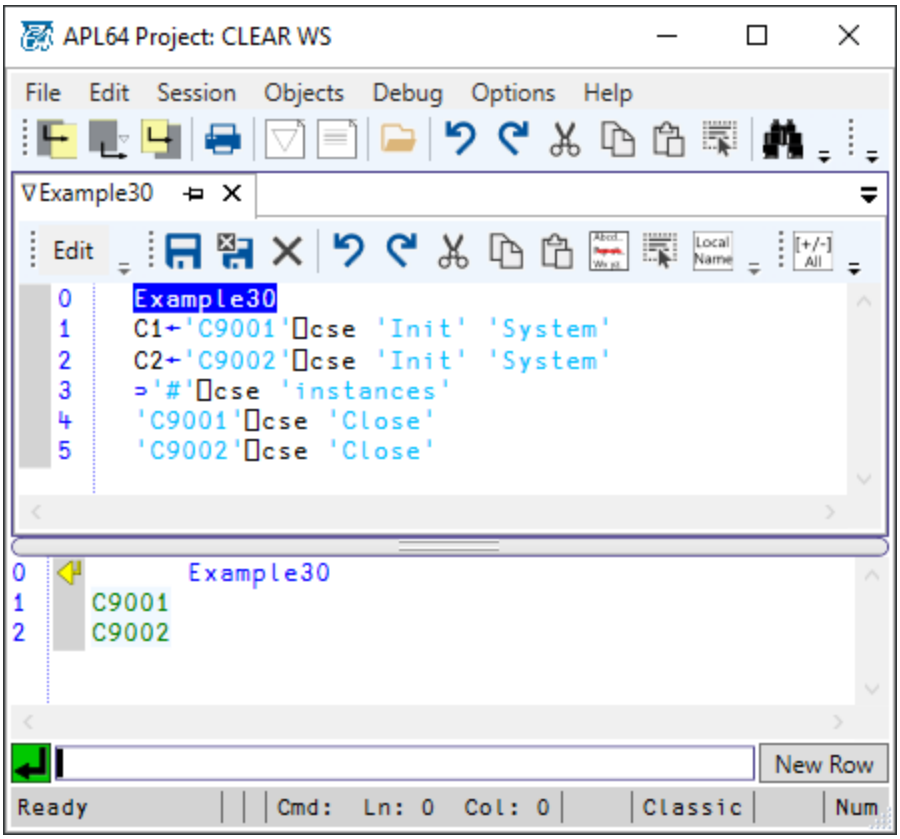
CSE System Object Properties

CSE System Object instances property

The CSE ‘instances’ property returns a vector of text vectors containing the names of the CSE instances which exist in the current APL64 session, if any.

Example #30:

```
Example30
C1←'C9001'□cse 'Init' 'System'
C2←'C9002'□cse 'Init' 'System'
⊃'#'□cse 'instances'
'C9001'□cse 'Close'
'C9002'□cse 'Close'
```

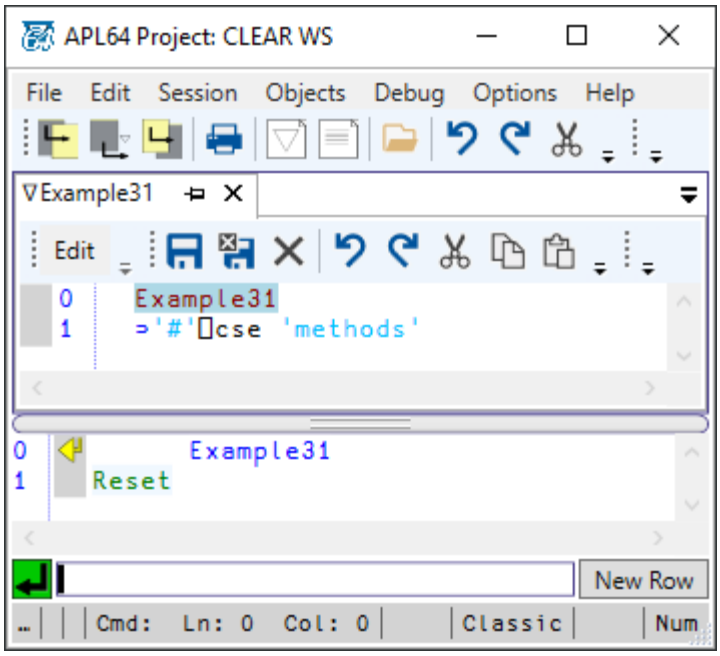


CSE System Object methods property

The CSE 'methods' property returns a vector of text vectors containing the names of the CSE System Object methods.

Example #31

```
Example31
⊃'#'⎕cse 'methods'
```

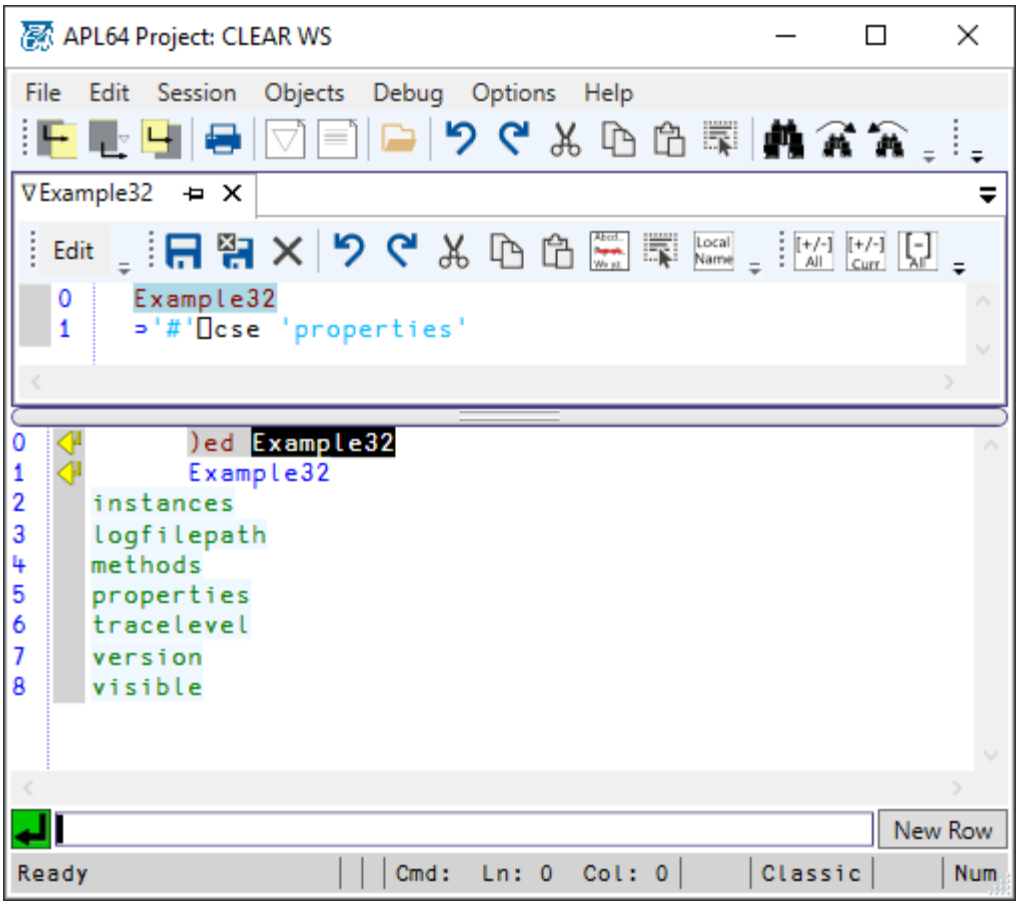


CSE System Object properties property

The CSE ‘properties’ property returns a vector of text vectors containing the names of the CSE System Object properties.

Example#32

```
Example32
⍝'#'⎕cse 'properties'
```



CSE System Object visible property

This property has been deprecated because the CSE is operating system cross-platform compatible.

CSE Instances

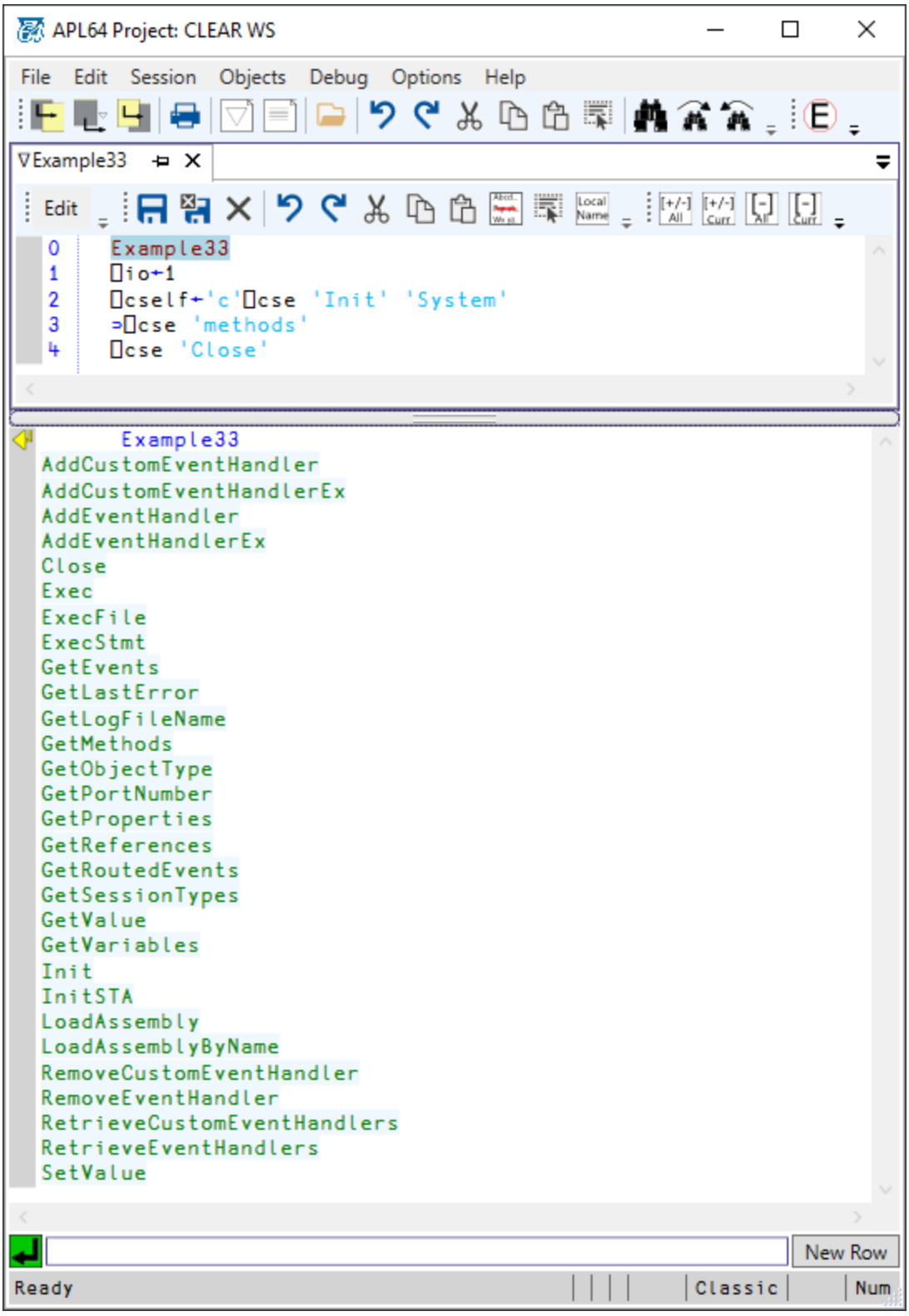
Within an APL64 instance one or more CSE instances may be created. Each CSE instance has its own collection APL programmer-defined .Net objects. Information may be passed between CSE instances by using the GetValue and SetValue CSE instance methods with the APL64 instance acting as the intermediary.

CSE Methods Applicable to a CSE Instance

The CSE 'methods' property returns a list of methods supported by an instance of the CSE. The action of a CSE method is always upon a specified instance of the CSE object. This specified CSE object may be explicitly named in the left argument to the □cse system function. The left argument to the □cse system function is optional and in that situation is assumed to be the current value of the CSE □cself system variable.

```
Example33
⎕io←1
```

```
□cself←'c'□cse 'Init' 'System'
⊃□cse 'methods'
□cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
Example33
Edit
0 Example33
1 ⎕io←1
2 ⎕cself←'c'⎕cse 'Init' 'System'
3 ⎕cse 'methods'
4 ⎕cse 'Close'
Example33
AddCustomEventHandler
AddCustomEventHandlerEx
AddEventHandler
AddEventHandlerEx
Close
Exec
ExecFile
ExecStmt
GetEvents
GetLastError
GetLogFileName
GetMethods
GetObjectType
GetPortNumber
GetProperties
GetReferences
GetRoutedEvents
GetSessionTypes
GetValue
GetVariables
Init
InitSTA
LoadAssembly
LoadAssemblyByName
RemoveCustomEventHandler
RemoveEventHandler
RetrieveCustomEventHandlers
RetrieveEventHandlers
SetValue
New Row
Ready
Classic
Num

CSE AddCustomEventHandler method

This method provides the means by which APL64 functions can be used to respond to .Net custom events created by the programmer. The APL64 calling environment may subscribe to such an event exposed by a C# object that exists in the state of the CSE object. Subscribing to such a C# event means that APL64 will be notified when the event fires. That notification to APL64 takes the form of the execution of a programmer-specified APL64 event handler function.

Custom events can be defined in .Net using the C# 'delegate' and 'event' statement prefixes. The programmer can design a custom event to fire when a class is constructed, a method is executed, when an exception occurs or when a some other action is performed, etc. The CSE 'AddCustomEventHandler' method provides custom event design flexibility because the programmer can define the .Net event arguments as a parameter list of simple APL64 value types.

The result of the CSE 'AddCustomEventHandlerMethod' is affected by the CSE 'returnonerror' property.

It is possible to have multiple APL64 event handler functions subscribed to the same C# event exposed by the same C# object. Event subscriptions are maintained by the CSE in 'first subscribed, first notified' order. It is also possible to have more than one C# event subscribed to the same APL64 event handler function.

The right argument of the CSE 'AddCustomEventHandler' method has four programmer-supplied elements.

CSE 'AddEventHandler' Method Right Argument	
Element #	**Description of CSE 'AddEventHandler' Method Right Argument Element**
1	String name of the existing C# object which exposes the event
2	String name of the C# custom event exposed by the specified C# object
3	String name of the APL64 event handler function
4	Programmer-defined argument to the APL64 event handler function or empty string

The CSE 'GetEvents' method may be used to determine the public events exposed by a C# object.

Because the .Net event being subscribed by APL64 is a custom event, the APL64 event handler function must have a specific syntax, e.g. MyEventHandler X:

- It must have no result
- It must have no left argument
- It must have a right argument

The programmer-defined argument to the APL64 event handler function:

- Is element #4 of the argument CSE 'AddCustomEventHandler' method
- Becomes element #4 of the right argument to the APL64 event handler function)
- Must be an APL64 text scalar, e.g. 'A' or text vector, e.g. 'ABC'
- Is passed to the programmer-defined APL64 event handler function as a text scalar or vector
- May be an empty text vector, i.e. ''.

When the APL64 event handler function is executed by the CSE because the specific C# object fired the subscribed event:

- The CSE will set the value of the APL64 event handler function right argument the programmerdefined (string) argument to the APL64 event handler function or empty string.
- The CSE will set the value of the APL64 □carg system variable to a vector with four elements:

Element #	□carg element while the APL64 event handler function is executing
1	CSE instance name
2	Name of the C# object which fired the subscribed event
3	Name of the C# event fired by the C# object
4	APL64 object containing the custom 'event args' values

- The APL64 event handler function, which is executed by the CSE when the subscribed C# object event fires, can use the CSE from within the handler function to:
- Use the APL64 □carg system variable to obtain information about the .Net custom event which fired.
- Modify the state of the applicable CSE object, i.e. obtain and set C# properties and use C# methods.

Because the event subscribed using the CSE 'AddCustomEventHandler' method is a custom event, the 'event args' are the programmer-defined arguments of the C# 'public delegate' executable statement which defined the custom event handler method syntax. The APL64 representation of these 'event args' are available as the 4th element of the APL64 system variable □carg during the execution of the APL64 event handler function subscribed to the .Net event. The data types of .Net objects which can be represented as elements of the APL64 system variable □carg are the same as those .Net types for the CSE 'GetValue' and 'SetValue' methods.

Since the values provided by the C# custom event will be available in the 4th element of the APL64 system variable □carg, it is necessary for the programmer to define the C# 'public delegate' for the .Net custom event with consideration of the data type limitations of APL. A CSE exception will occur if the programmer defines an argument of the C# 'public delegate' that is a .Net type which has no representation in APL.

If a .Net event is already associated with a .Net object, it is not necessary for the programmer to create a .Net custom event. Instead the pre-existing .Net event can be handled as a .Net routed event using the CSE 'AddEventHandler' or 'AddEventEventHandlerEx' methods.

Example 180:
In this example a C# class is defined containing a method which when executed will fire a .Net custom event. APL64 will subscribe to this event and an APL64 event handler function will be executed when this custom event fires.

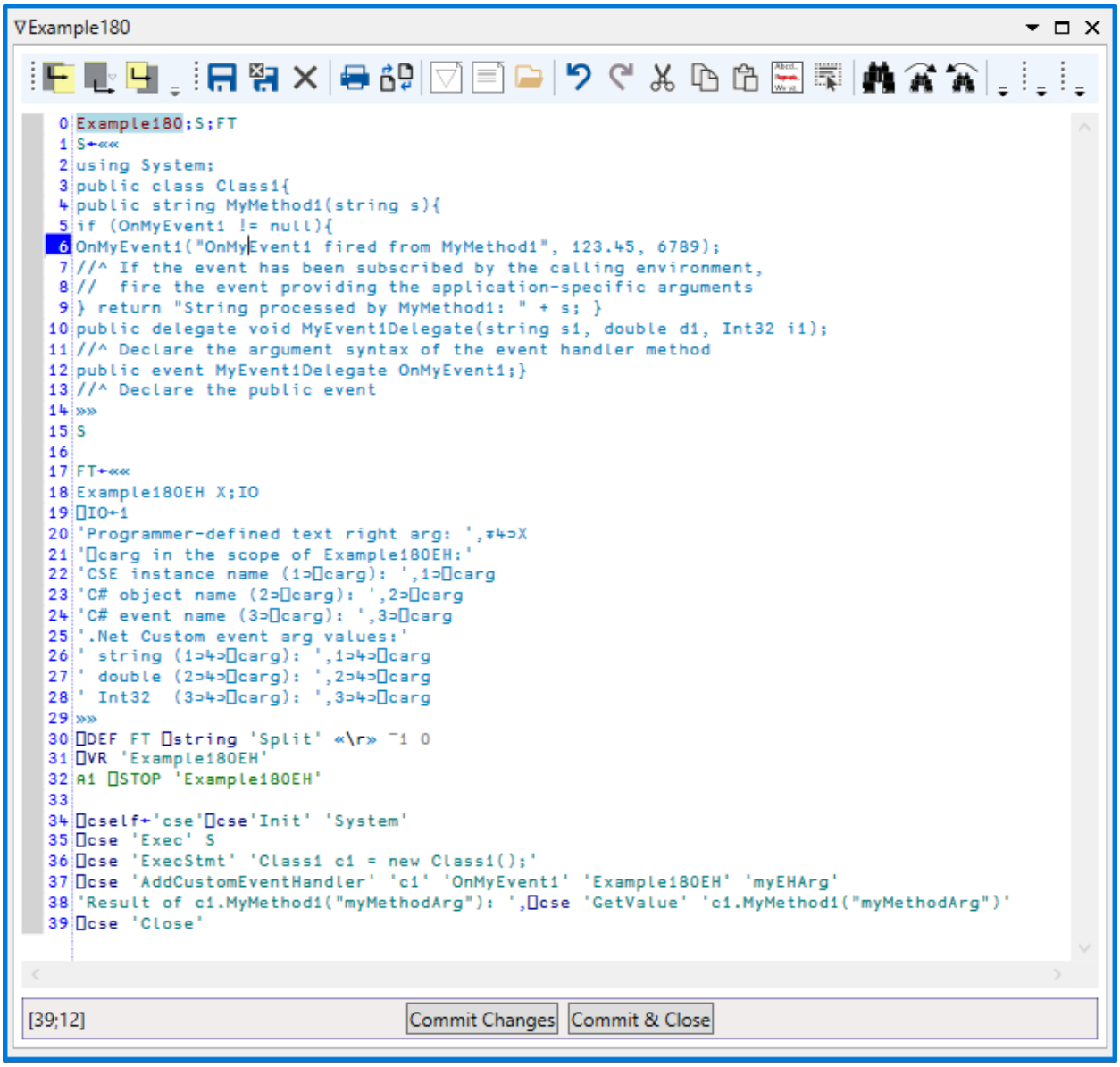
```
Example180;S;FT
S←««
using System;
public class Class1{
public string MyMethod1(string s){
if (OnMyEvent1 != null){
OnMyEvent1("OnMyEvent1 fired from MyMethod1", 123.45, 6789);
//^ If the event has been subscribed by the calling environment,
//  fire the event providing the application-specific arguments
} return "String processed by MyMethod1: " + s; }
public delegate void MyEvent1Delegate(string s1, double d1, Int32 i1);
//^ Declare the argument syntax of the event handler method
public event MyEvent1Delegate OnMyEvent1;}
//^ Declare the public event
»»
S

FT←««
Example180EH X;IO
⎕IO←1
'Programmer-defined text right arg: ',⍕4⊃X
'⎕carg in the scope of Example180EH:'
'CSE instance name (1⊃⎕carg): ',1⊃⎕carg
'C# object name (2⊃⎕carg): ',2⊃⎕carg
'C# event name (3⊃⎕carg): ',3⊃⎕carg
'.Net Custom event arg values:'
' string (1⊃4⊃⎕carg): ',1⊃4⊃⎕carg
' double (2⊃4⊃⎕carg): ',2⊃4⊃⎕carg
' Int32  (3⊃4⊃⎕carg): ',3⊃4⊃⎕carg
»»
⎕DEF FT ⎕string 'Split' «\r» ¯1 0
⎕VR 'Example180EH'
⍝1 ⎕STOP 'Example180EH'

⎕cselF←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
⎕cse 'ExecStmt' 'Class1 c1 = new Class1();'
⎕cse 'AddCustomEventHandler' 'c1' 'OnMyEvent1' 'Example180EH' 'myEHArg'
'Result of c1.MyMethod1("myMethodArg"): ',⎕cse 'GetValue' 'c1.MyMethod1("myMethodArg")'
⎕cse 'Close'
```

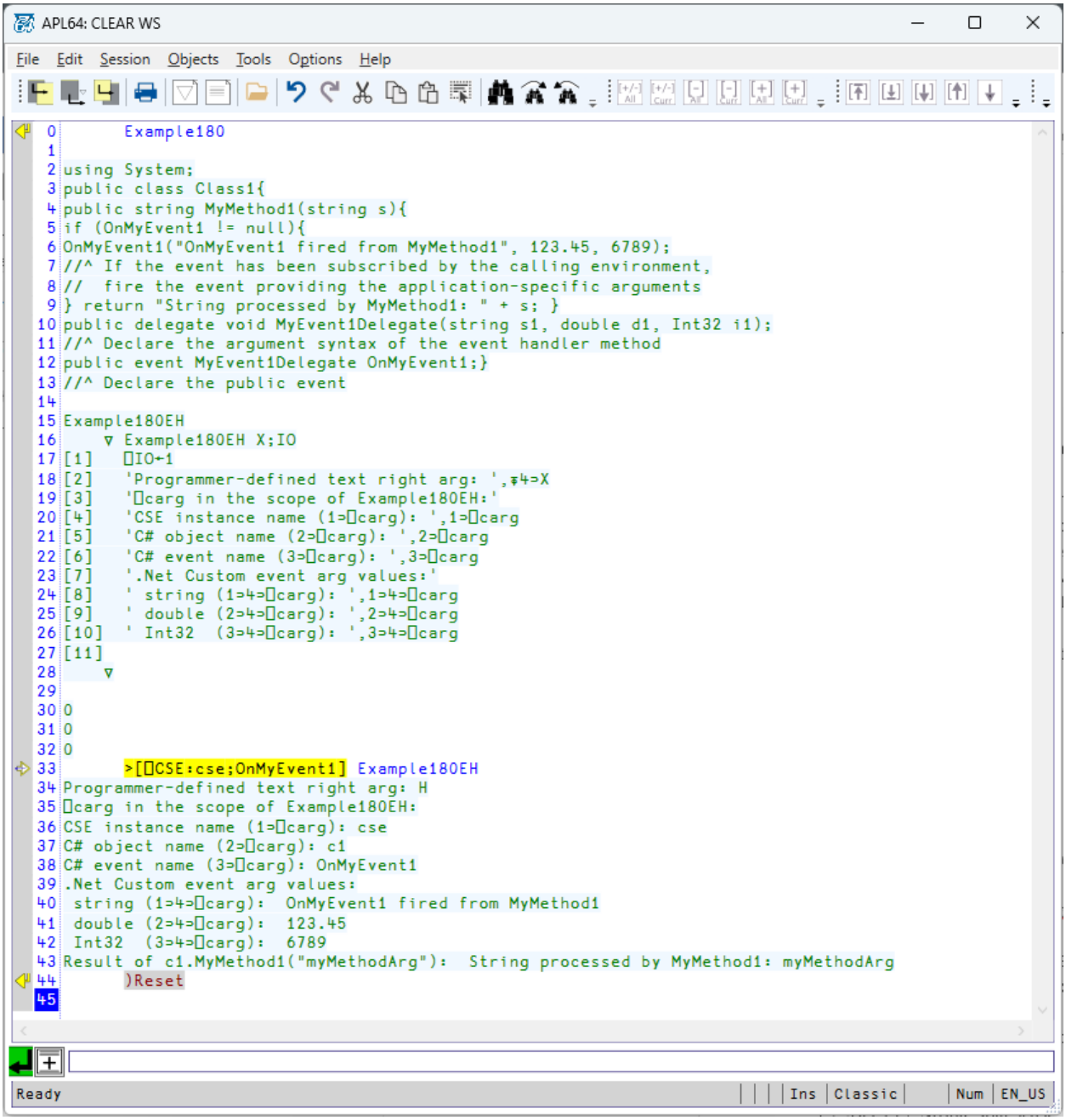
The APL64 event handler function performs the programmer-defined actions when the custom event fires.

The Example180 function:

- Creates a CSE instance
- Defines and executes the C# script which
    - Defines the .Net class ‘Class1’
    - Defines the .Net public event ‘OnMyEvent1’
    - Defines the .Net public delegate ‘OnMyEvent1Delegate’ for the event ‘OnMyEvent1’ with ‘event args’ with .Net data types string, double and Int32 which have analogous APL64 data types text vec tor, floating point scalar and integer scalar
    - Defines the .Net method ‘MyMethod1’ which fires the .Net event ‘OnMyEvent1’
- Defines the Example180EH function which is executed when the MyEvent fires.
- Creates an instance of the Class1 class
- Executes the Class1.MyMethod1 function which causes MyEvent to fire
- Displays the result of the Class1.MyMethod1 function
- Closes the CSE instance

Remove the remark to activate the stop to examine the APL state when the event handler is executed.


∇Example180
Example180;S;FT
S←⍬⍬
using System;
public class Class1{
public string MyMethod1(string s){
if (OnMyEvent1 != null){
OnMyEvent1("OnMyEvent1 fired from MyMethod1", 123.45, 6789);
//^ If the event has been subscribed by the calling environment,
// fire the event providing the application-specific arguments
} return "String processed by MyMethod1: " + s; }
public delegate void MyEvent1Delegate(string s1, double d1, Int32 i1);
//^ Declare the argument syntax of the event handler method
public event MyEvent1Delegate OnMyEvent1;}
//^ Declare the public event
⍬⍬⍬
S
FT←⍬⍬
Example180EH X;IO
⎕IO←1
'Programmer-defined text right arg: ',⍕4⊃X
'⎕carg in the scope of Example180EH:'
'CSE instance name (1⊃⎕carg): ',1⊃⎕carg
'C# object name (2⊃⎕carg): ',2⊃⎕carg
'C# event name (3⊃⎕carg): ',3⊃⎕carg
'.Net Custom event arg values:'
' string (1⊃4⊃⎕carg): ',1⊃4⊃⎕carg
' double (2⊃4⊃⎕carg): ',2⊃4⊃⎕carg
' Int32 (3⊃4⊃⎕carg): ',3⊃4⊃⎕carg
⍬⍬⍬
⎕DEF FT ⎕string 'Split' ⍬⍬⍬«\r» ¯1 0
⎕VR 'Example180EH'
⍝1 ⎕STOP 'Example180EH'
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
⎕cse 'ExecStmt' 'Class1 c1 = new Class1();'
⎕cse 'AddCustomEventHandler' 'c1' 'OnMyEvent1' 'Example180EH' 'myEHArg'
'Result of c1.MyMethod1("myMethodArg"): ',⎕cse 'GetValue' 'c1.MyMethod1("myMethodArg")'
⎕cse 'Close'
[39;12]
Commit Changes
Commit & Close



Example #185

This example illustrates that the value of ⎕carg is local to the scope of the APL64 function handling the .Net custom event. In this example, APL64 subscribes to a .Net custom event and the associated APL64 event handler function then subscribes to a second .Net custom event.

- Creates an instance of the CSE
- Defines and executes the CSE script, S1:
  - Defines a .Net public class ‘Class1’
  - Defines the .Net public event ‘OnMyEvent1’ in this class

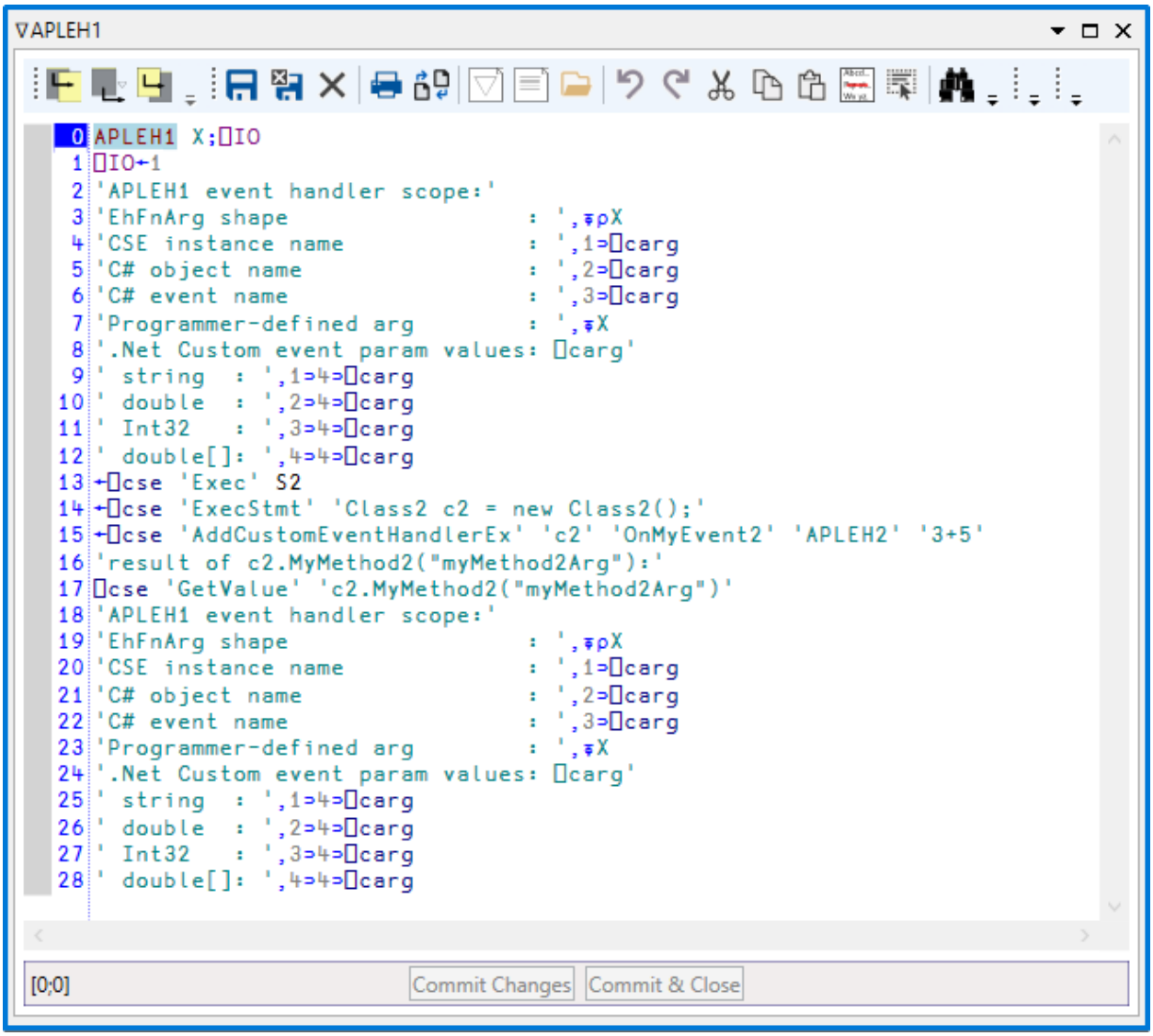
    - Defines a .Net public delegate 'MyEvent1Delegate' for the 'OnMyEvent1' event which provides for four 'event args' of varying data type (string, double, Int32 and double[]), each of which have valid representations in APL64 data types (text vector, floating point scalar, integer scalar and vector of floating point)..
    - Defines a .Net public method 'MyMethod1' in this class which fires the .Net event 'OnMyEvent1'
- Defines the CSE script, S2:
    - Defines a .Net public class 'Class2'
    - Defines the .Net public event 'OnMyEvent2' in this class
    - Defines a .Net public delegate 'MyEvent2Delegate' for the 'OnMyEvent2' event
    - Defines a .Net public method 'MyMethod2' in this class which fires the .Net event 'OnMyEvent2'
- Creates an instance of the .Net 'Class1' class called 'c1'
- Subscribes to the .Net custom event 'OnMyEvent1' with the APL64 event handler function 'APLEH1' function which runs when the .Net event 'OnMyEvent1' fires
- Obtains the value of the result of the C# method 'MyMethod1()' of the .Net class 'Class1'
- Closes the CSE instance

Define the APL64 event handler function APLEH1:

- Runs when the .Net custom event 'OnMyEvent1' fires
- Has no left argument
- Displays the value of its right argument which contains the programmer-defined (string-values) argument specified when the CSE 'AddCustomEventHandler' method was executed
- Displays the values of the APL64 □carg system variable provided by the CSE when the .Net event fired, including the 4th element of □carg which contains the .Net custom 'event args' values defined in the C# delegate statement in the CSE 'Script1' script
- Executes the CSE script, S2
- Creates an instance of the .Net class 'Class2'
- Subscribes the APL64 event handler function 'APLEH2' to the .Net custom event 'OnMyEvent2' in 'Class2'
- Obtains the value of the result of the C# method 'MyMethod2()' of the .Net class 'Class2'
- Displays its right argument value and the values of the CSE □carg system variable. Since APL64 is 'single-threaded', the firing of the .Net event 'OnMyEvent2' and the running of the APL64 event handler function 'APLEH2' will occur before the 'OnMyEvent1' values are displayed again.

```
APLEH1 X;⎕IO
⎕IO←1
'APLEH1 event handler scope:'
'EhFnArg shape          : ',⍕⍴X
'CSE instance name      : ',1⊃⎕carg
'C# object name         : ',2⊃⎕carg
'C# event name          : ',3⊃⎕carg
```

```
'Programmer-defined arg       : ', ⍕X
'.Net Custom event param values: ⎕carg'
' string  : ',1⊃4⊃⎕carg
' double  : ',2⊃4⊃⎕carg
' Int32   : ',3⊃4⊃⎕carg
' double[]: ',4⊃4⊃⎕carg
←⎕cse 'Exec' S2
←⎕cse 'ExecStmt' 'Class2 c2 = new Class2();'
←⎕cse 'AddCustomEventHandlerEx' 'c2' 'OnMyEvent2' 'APLEH2' '3+5'
'result of c2.MyMethod2("myMethod2Arg"):'
⎕cse 'GetValue' 'c2.MyMethod2("myMethod2Arg")'
'APLEH1 event handler scope:'
'EhFnArg shape              : ', ⍕ρX
'CSE instance name          : ',1⊃⎕carg
'C# object name             : ',2⊃⎕carg
'C# event name              : ',3⊃⎕carg
'Programmer-defined arg       : ', ⍕X
'.Net Custom event param values: ⎕carg'
' string  : ',1⊃4⊃⎕carg
' double  : ',2⊃4⊃⎕carg
' Int32   : ',3⊃4⊃⎕carg
' double[]: ',4⊃4⊃⎕carg
```

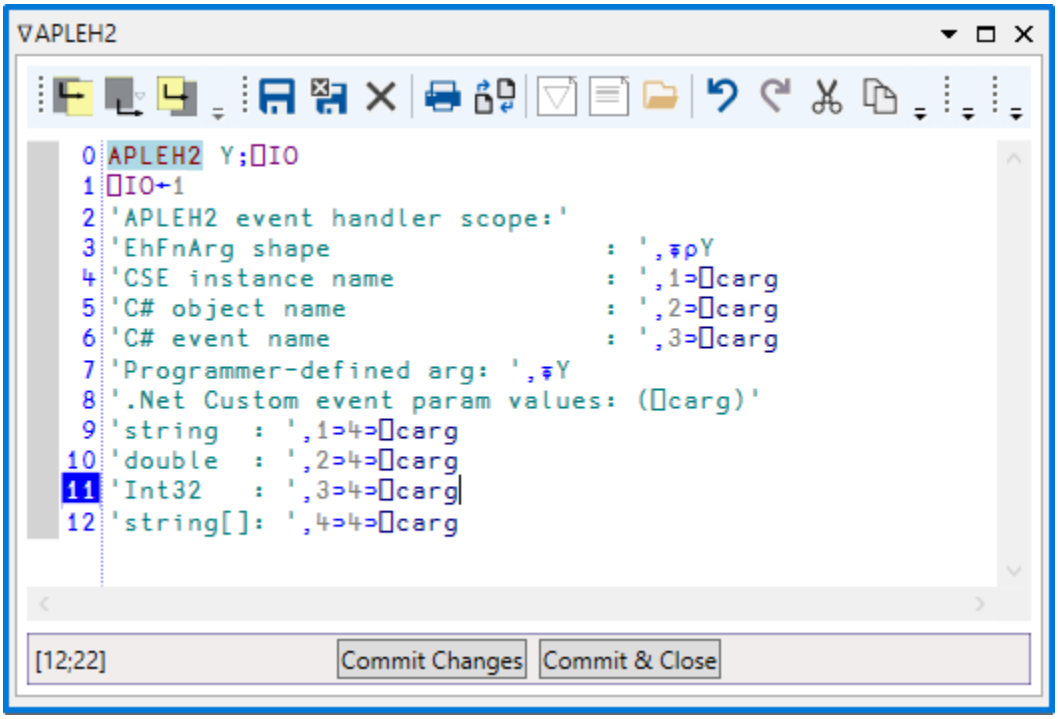


Define the APL64 event handler function APLEH2 which is subscribed to the .Net event 'OnMyEvent2':

- Runs when the .Net event 'OnMyEvent2' fires
- Has no left argument
- Displays the values of its right argument and the APL64 system variable □carg

```
APLEH2 Y;□IO
□IO←1
'APLEH2 event handler scope:'
'EhFnArg shape          : ',⍕⍴Y
'CSE instance name      : ',1⊃□carg
'C# object name         : ',2⊃□carg
'C# event name          : ',3⊃□carg
'Programmer-defined arg: ',⍕Y
```

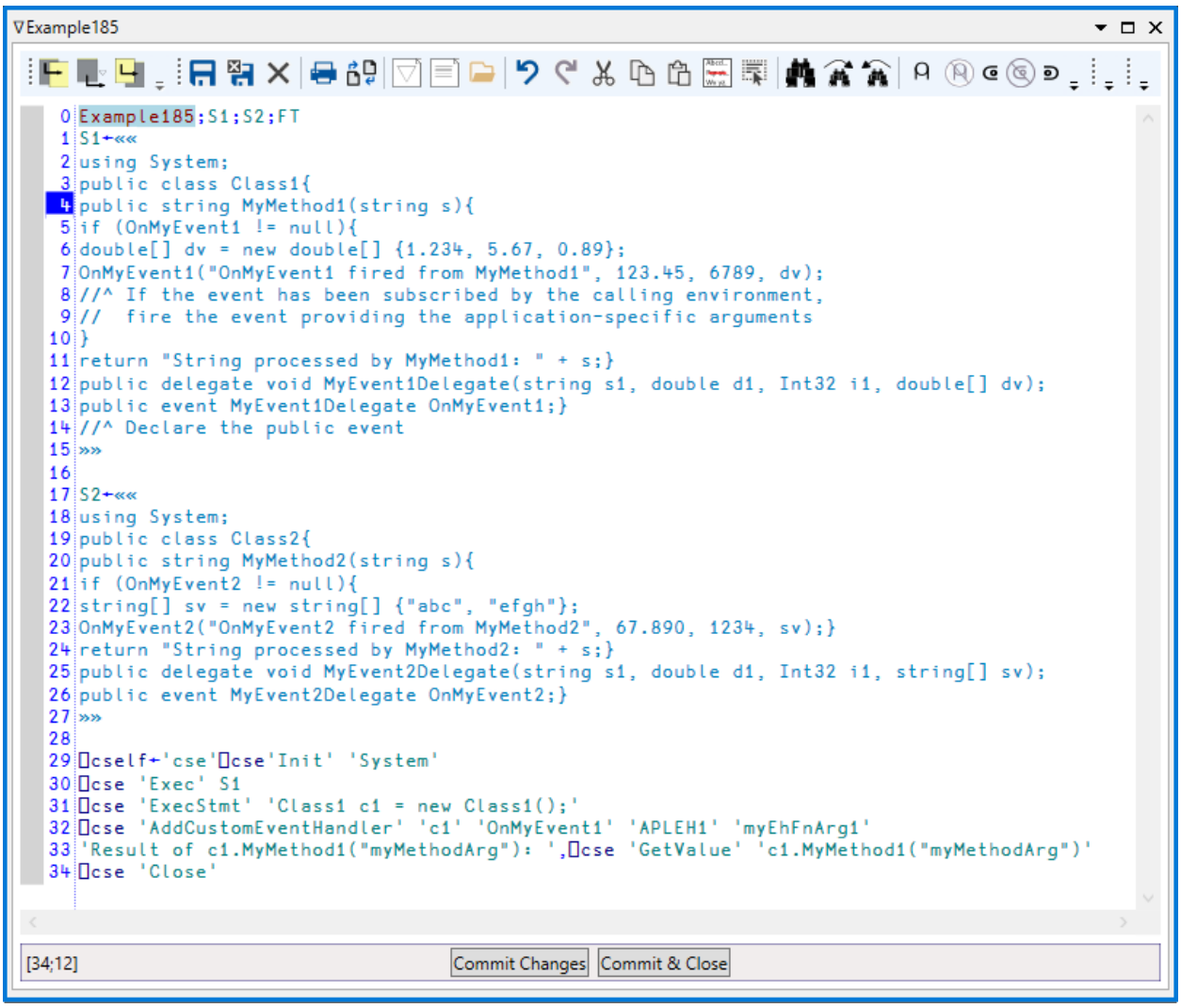
```
'.Net Custom event param values: (⎕carg)'
'string  : ',1⊃4⊃⎕carg
'double  : ',2⊃4⊃⎕carg
'Int32   : ',3⊃4⊃⎕carg
'string[]: ',4⊃4⊃⎕carg
```



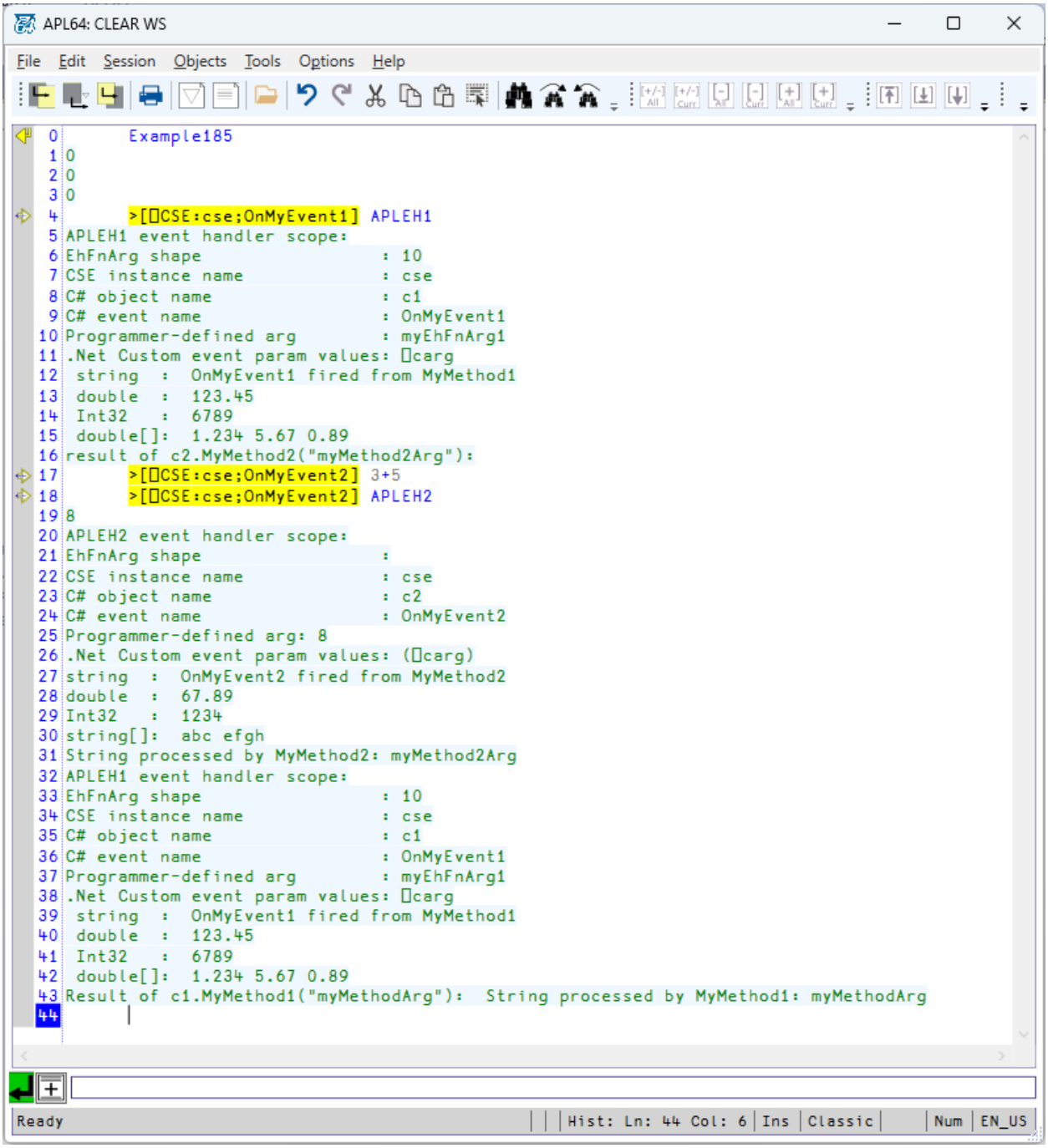
Define the Example185 function

```
Example185;S1;S2;FT
S1←««
using System;
public class Class1{
public string MyMethod1(string s){
if (OnMyEvent1 != null){
double[] dv = new double[] {1.234, 5.67, 0.89};
OnMyEvent1("OnMyEvent1 fired from MyMethod1", 123.45, 6789, dv);
//^ If the event has been subscribed by the calling environment,
//  fire the event providing the application-specific arguments
}
return "String processed by MyMethod1: " + s;}
public delegate void MyEvent1Delegate(string s1, double d1, Int32 i1, double[] dv);
public event MyEvent1Delegate OnMyEvent1;}
//^ Declare the public event
```

```
»»

S2←««
using System;
public class Class2{
public string MyMethod2(string s){
if (OnMyEvent2 != null){
string[] sv = new string[] {"abc", "efgh"};
OnMyEvent2("OnMyEvent2 fired from MyMethod2", 67.890, 1234, sv);}
return "String processed by MyMethod2: " + s;}
public delegate void MyEvent2Delegate(string s1, double d1, Int32 i1, string[] sv);
public event MyEvent2Delegate OnMyEvent2;}
»»

□cself←'cse'□cse'Init' 'System'
□cse 'Exec' S1
□cse 'ExecStmt' 'Class1 c1 = new Class1();'
□cse 'AddCustomEventHandler' 'c1' 'OnMyEvent1' 'APLEH1' 'myEhFnArg1'
'Result of c1.MyMethod1("myMethodArg"): ',□cse 'GetValue' 'c1.MyMethod1("myMethodArg")'
□cse 'Close'
```



Run the Example185 function:



CSE AddEventHandler Method

This method provides the means by which APL64 functions can be used to respond to actions on C# objects which fire .Net routed events. The APL64 calling environment may subscribe to such an event exposed by a C# object that exists in the state of the CSE object. Subscribing to such a C# event means that APL64 will be notified when the event fires. That notification to APL64 takes the form of the execution of a programmer-specified APL64 event handler function.

The result of the CSE 'AddEventHandlerMethod' is affected by the CSE '<u>returnonerror</u>' property.

The right argument of the CSE 'AddEventHandler' method has four programmer-supplied elements. When APL64 subscribes to a .Net routed event using the CSE 'AddEventHandler' method the CSE will record these elements so that if and when the C# object fires the specified event, the CSE will execute the APL64 event handler function.

CSE 'AddEventHandler' Method Right Argument	
Element #	**Description of CSE 'AddEventHandler' Method Right Argument Element**
1	String name of the existing C# object which exposes the event
2	String name of the C# routed event exposed by the specified C# object
3	String name of the APL64 event handler function
4	Programmer-defined argument to the APL64 event handler function

The CSE 'GetEvents' method may be used to determine the public events exposed by a C# object.

Because the .Net event being subscribed by APL64 is a routed event, the APL64 event handler function must have a specific syntax, e.g. MyEventHandler X:

- It must have no result
- It must have no left argument
- It must have a right argument

When the APL64 event handler function is executed by the CSE because the specific C# object fired the subscribed event, the CSE will set the value of the APL64 event handler function right argument to an APL64 vector:

APL64 Event Handler Right Argument Elements	
Element #	Description of the APL64 Right Argument Element
1	Name of the instance of the associated CSE object
2	Name of the C# object which fired the subscribed event
3	Name of the C# event fired by the C# object
4	Key in the _cseEvt dictionary to the C# event args object
5	2-column array of event argument property names and values
6	Programmer-defined argument to the event handler function

The programmer-defined argument to the APL64 event handler function:

- Is element #4 of the argument CSE 'AddEventHandler' method
- Becomes Element #6 of the right argument to the APL64 event handler function)
- Must be an APL64 text scalar, e.g. 'A' or text vector, e.g. 'ABC'
- Is passed to the programmer-defined APL64 event handler function as a text scalar or vector
- May be an empty text vector, i.e. ''.

The APL64 event handler function, which is executed by the CSE when the subscribed C# object event fires, can use the CSE from within the handler function to modify the state of the applicable CSE object, i.e. obtain and set C# properties and use C# methods.

The number of elements assigned to the required right argument of an APL64 CSE event handler function may increase in a future version of APL64.

It is possible to have multiple APL64 event handler functions subscribed to the same C# event exposed by the same C# object. Event subscriptions are maintained by the CSE in 'first subscribed, first notified' order. It is also possible to have more than one C# event subscribed to the same APL64 event handler function.

Accessing the .Net 'event args' object from within the APL64 event handler function:

- The property values available to an APL64 event handler function are called the 'event args' object in .Net. The 'event args' object can vary depending on the event type which has been subscribed. This is because in .Net it is possible to define a custom 'event args' class which inherits the properties of the base 'event args' class but also adds additional custom 'event args' properties.
- It is also possible for an 'event args' property to have a .Net data type which has no representation in APL64.
- To properly support these features of .Net 'event args' objects, the CSE provides two ways for the programmer to access the 'event args' from within the APL64 event handler function.
    - The 5th element provided by the CSE in the right argument of the APL64 event handler function is a two column array with one row for each 'event args' property. The entries in this array will vary with the .Net object which fired the event. The 1st column provides the name of the 'event args' property and the 2nd column provides:
        - The value of that 'event args' property if it has an APL64 representation, or
        - A string ('See _cseEvt for this object') indicating that, because it cannot be represented directly in APL64, its value must be obtained by using the '_cseEvt' dictionary.
    - The 4th element provided by the CSE in the right argument of the APL64 event handler function is a string key to the C# 'event args' object in the '_cseEvt' dictionary. Using this object and CSE methods all properties of the 'event args' object can be accessed from within the APL64 event handler function. The .Net 'event args' object is present in the '_cseEvt' dictionary only when the APL64 event handler function is executing. Using the 4th element, the APL64 programmer will have access to all the members of the C# 'event args' object, even if such a member is not directly representable in the simple value types available in APL64.
- Whereas the CSE 'AddCustomEventHandler' and 'AddCustomEventHandlerEx' methods are limited to .Net custom 'event args' values which have well-defined APL64 representations, the CSE 'AddEventHandler' and 'AddEventHandlerEx' methods provide access to .Net routed 'event args' values of any .Net type even if they do not have APL64 representations.
- Additional documentation is available about accessing the C# event args object from APL64 using the CSE.

CSE AddCustomEventHandlerEx method

The action of the CSE 'AddCustomEventHandlerEx' method is the same as that of the CSE 'AddCustomEventHandler' method except:

- The 4th argument to the CSE 'AddCustomEventHandlerEx' method will be treated as an APL64 executable statement, whereas the CSE 'AddCustomEventHandler' method treats it as a text vector.
- The value resulting from the execution by the APL64 interpreter of that executable statement will be passed to the APL64 event handler function the value of its right argument when the .Net event subscribed by the CSE 'AddCustomEventHandlerEx' method fires.

CSE AddEventHandlerEx method

The action of the CSE 'AddEventHandlerEx' method is the same as that of the CSE 'AddEventHandler' method except:

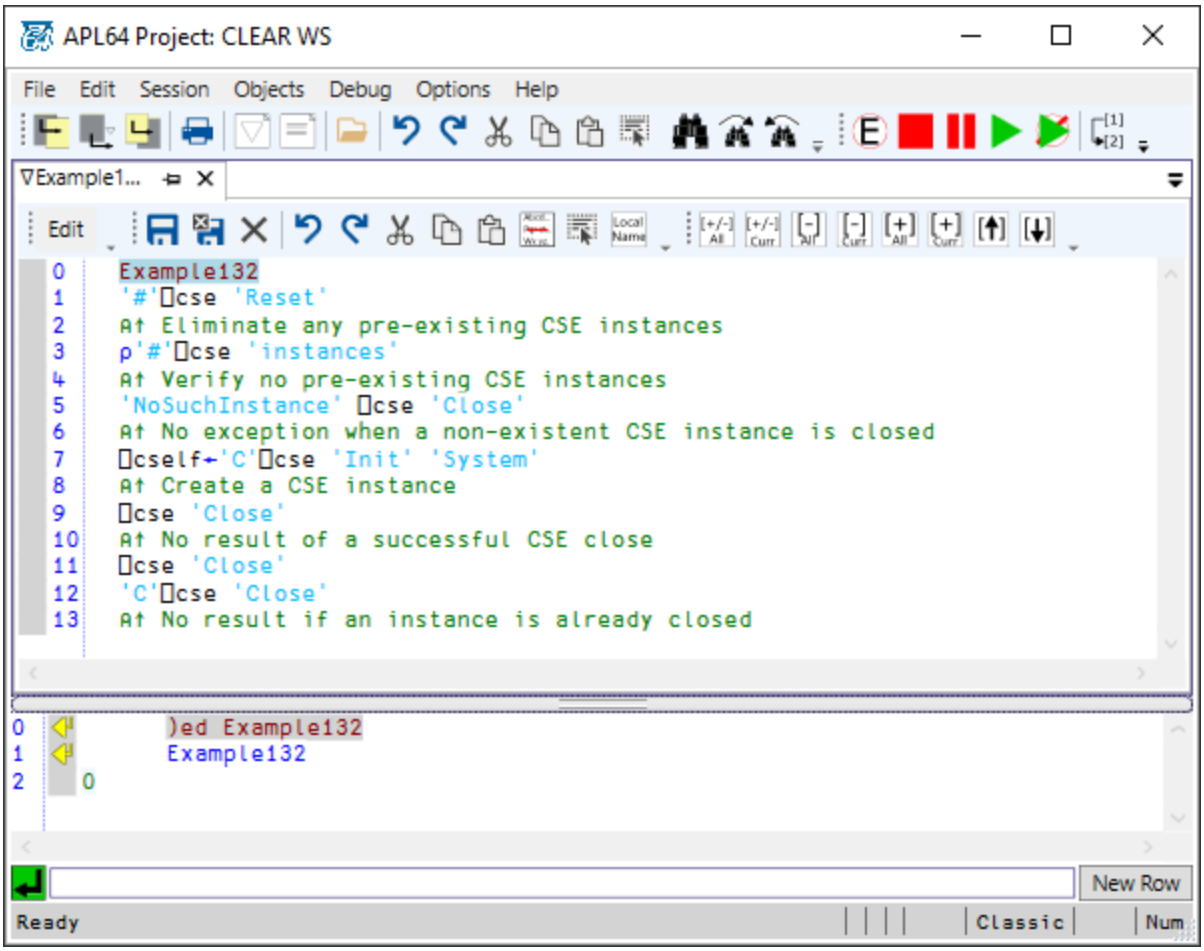
- The 4th argument to the CSE 'AddEventHandlerEx' method will be treated as an APL64 executable statement, whereas the CSE 'AddEventHandler' method treats it as a text vector.
- The value resulting from the execution by the APL64 interpreter of that executable statement will be passed to the APL64 event handler function as the 6th element of its right argument when the .Net event subscribed by the CSE 'AddEventHandlerEx' method fires.
- Since the 4th argument to the CSE 'AddEventHandlerEx' method is executed by APL64 it is possible to pass any APL64 object to the APL64 event handler function.

CSE Close Method

The 'Close' method will dispose of the specified instance of the C# script engine. If the CSE 'Close' method is executed on an instance of the CSE which is already closed or does not exist, no error will be thrown.

Example #132

```
Example132
'#'□cse 'Reset'
⍝↑ Eliminate any pre-existing CSE instances
⍴'#'□cse 'instances'
⍝↑ Verify no pre-existing CSE instances
'NoSuchInstance' □cse 'Close'
⍝↑ No exception when a non-existent CSE instance is closed
□cself←'C'□cse 'Init' 'System'
⍝↑ Create a CSE instance
□cse 'Close'
⍝↑ No result of a successful CSE close
□cse 'Close'
'C'□cse 'Close'
⍝↑ No result if an instance is already closed
```

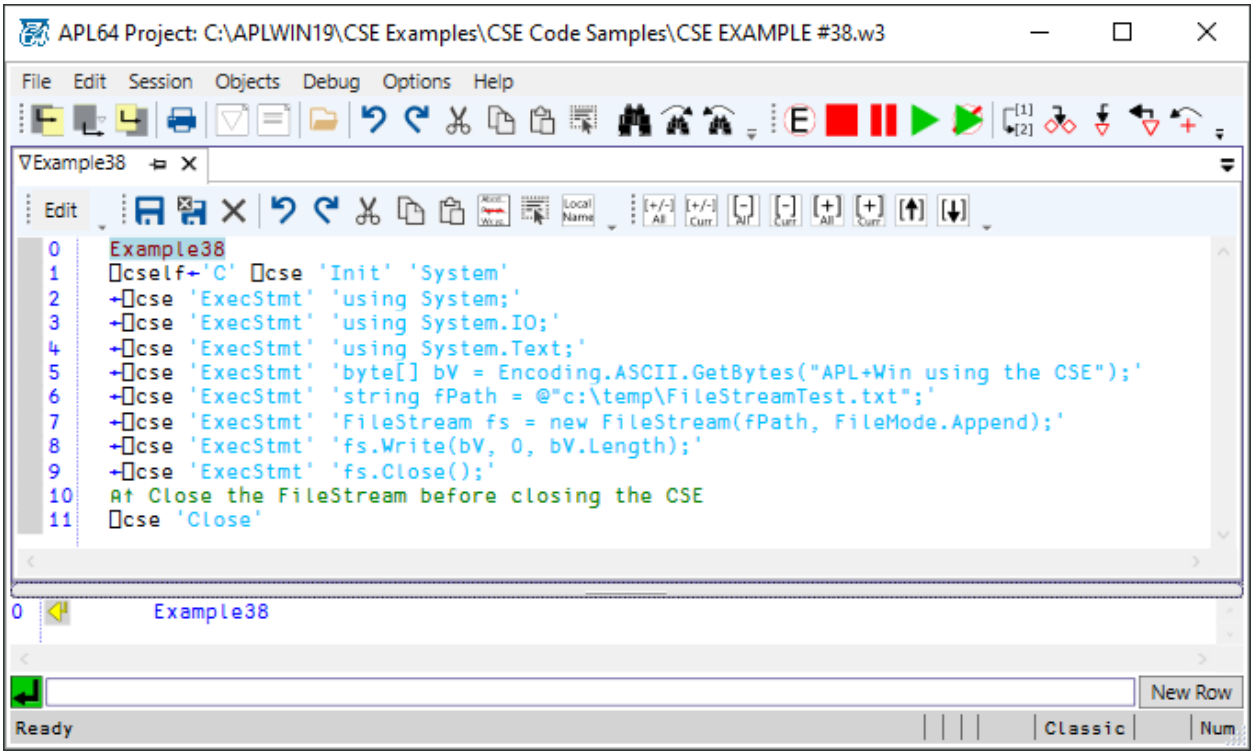


Example #38:
Considering the design of the .Net environment, all possible resources and assemblies used by the closed CSE instance will be disposed. When using C# to create instances of certain .Net objects, care must be taken to dispose of them before using the CSE 'Close' method. Otherwise 'orphan' instances of .Net objects may remain after the CSE 'Close' method is used. This is not a limitation of the CSE 'Close' method, but is instead inherent in .Net.

For example, when using C# to create an instance of a Microsoft Office object, e.g. Excel Application, using C# to access a database using an ADO.Net connection object or using a .Net file stream, it is important to dispose of these .Net objects before using the CSE 'Close' method. An alternative to an explicit dispose of a .Net object, the using (…){…} syntax may be used.

```
Example38
⎕cself←'C' ⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
←⎕cse 'ExecStmt' 'using System.IO;'
←⎕cse 'ExecStmt' 'using System.Text;'
←⎕cse 'ExecStmt' 'byte[] bV = Encoding.ASCII.GetBytes("APL+Win using the CSE");'
```

```
←⎕cse 'ExecStmt' 'string fPath = @"c:\temp\FileStreamTest.txt";'
←⎕cse 'ExecStmt' 'FileStream fs = new FileStream(fPath, FileMode.Append);'
←⎕cse 'ExecStmt' 'fs.Write(bV, 0, bV.Length);'
←⎕cse 'ExecStmt' 'fs.Close();'
⍝↑ Close the FileStream before closing the CSE
⎕cse 'Close'
```



CSE Exec Method

The 'Exec' method causes the CSE instance to execute the CSE script specified in the required right argument containing the text of the C# script.

The Exec method right argument can be a character vector, character matrix or scalar string. For successful operation the C# script must contain none, one or a collection of valid C# statements. The validity of the C# statement is verified using the Microsoft C# debugger within the existing state of the CSE instance.

The result of the CSE 'Exec' is affected by the value of the CSE 'returnonerror' property. When the CSE 'returnonerror' property value is 0, the result of the CSE 'Exec' method is a scalar integer:

- 0 indicates that no error was reported by the C# debugger and the script was successfully executed
- -1 indicates that there was an error reported by the C# debugger and the script was not successfully executed. Use the CSE 'GetLastError' method to obtain the text of the C# debugger error message. A

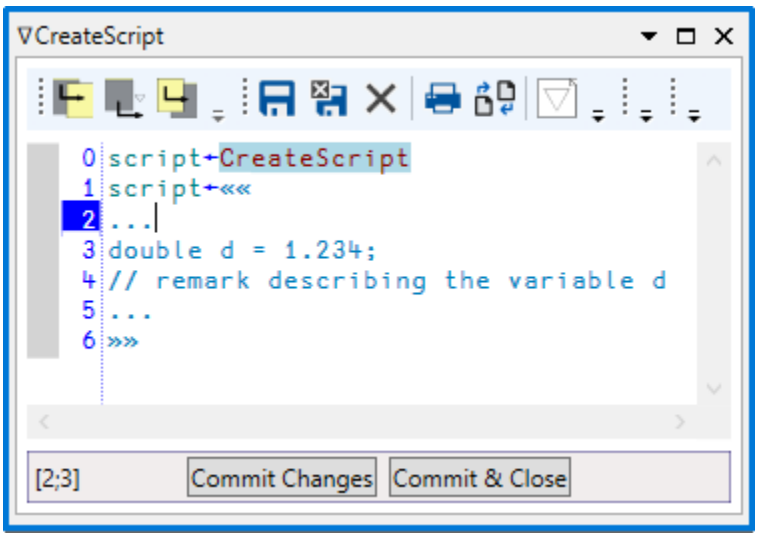
-1 result indicates that an error occurred within the .Net environment of the instance of the CSE object. In this case no APL64 exception is thrown and □dm is not modified to contain the error message reported by the C# debugger.

If an APL64 statement designed to use the CSE 'Exec' method contains a syntactical error detected by the APL64 debugger, an APL64 exception is thrown and □dm is updated to contain the error message reported by the APL64 debugger. APL64 does not parse contents of the CSE script argument to CSE 'Exec' method and so the APL64 debugger will not report errors in the C# statements contained in that CSE script using □dm. Errors, if any, are reported by the Microsoft C# debugger.

While the CSE script is being executed, execution control of the application is transferred to the .Net environment. Execution control will return to the APL64 environment which created the CSE object when the CSE script has completed execution. While the CSE script is executing it is possible to temporarily transfer execution control to that APL64 environment and then return execution control back to the CSE script by including an 'APL:' statement in the CSE script.

If a character vector or scalar string C# script or a row of a character matrix C# Script contains remarks, the remarks must be separated by newline characters from subsequent C# statements so the subsequent statements are effectively processed.

When creating a CSE script within an APL64 programmer-defined function, the APL64 string line continuation structure, ««...»», is convenient. Using this structure clarifies the script because there is no need to use concatenation and quotation marks to join script lines. When a line in the string script variable is continued on a subsequent line, an APL newline character, «\r» is included between previous line and the next line.



Example #248 CSE Exec Scripts as Multi-Line String Scalars

The APL64 CSE Exec action supports a multi-line string. This means that entering the CSE script as the right argument of the APL64 CSE Exec action is easy, does not required multiple APL concatenations, and is easy to read in the APL64 programmer-defined function creating the CSE script.

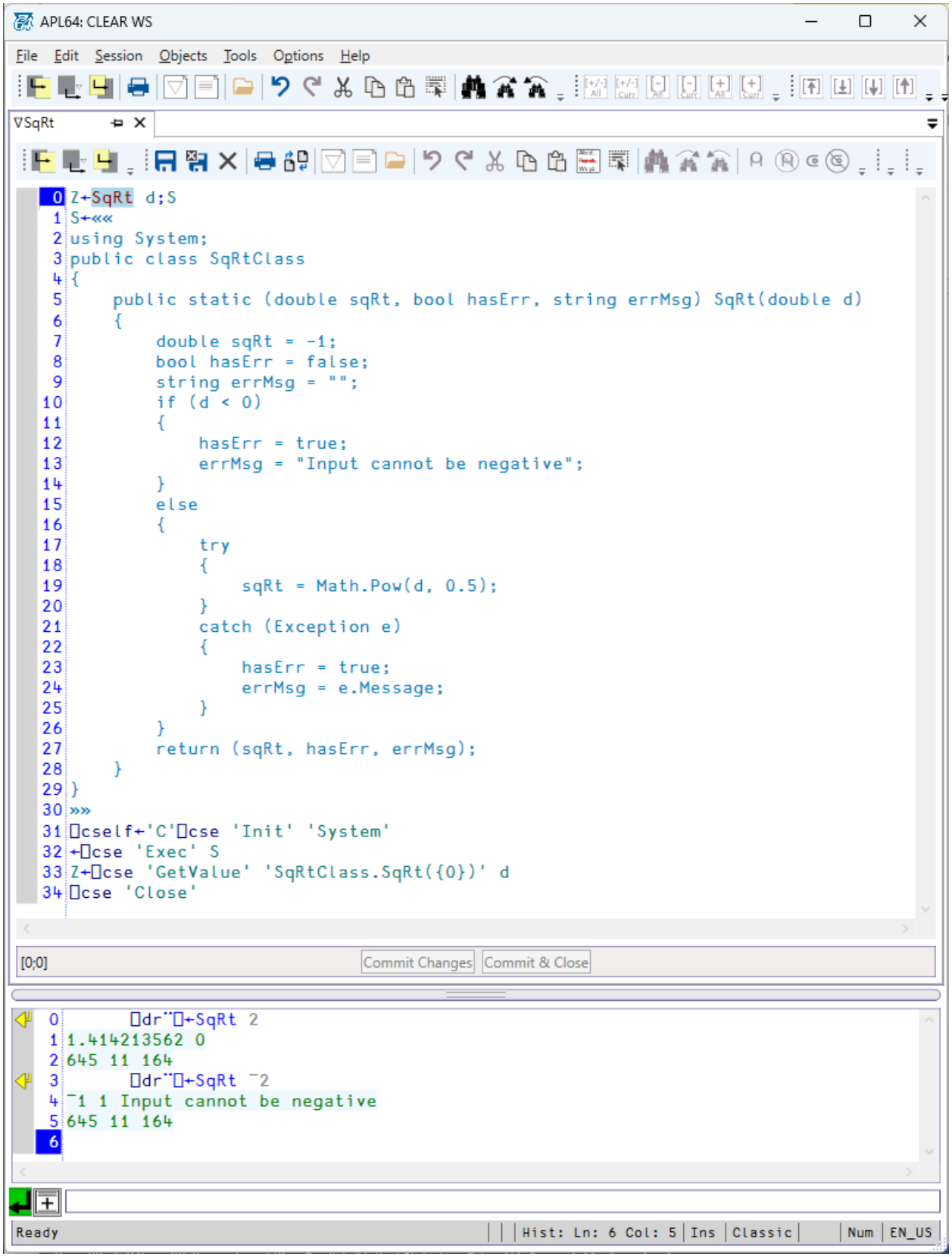
In APL64 multi-line string uses the line continuation structure for a string:
varName ← «« Text of string ... »»

Create the SqRt function using this definition:

```
Z←SqRt d;S
S←««
using System;
public class SqRtClass
{
   public static (double sqRt, bool hasErr, string errMsg) SqRt(double d)
   {
      double sqRt = -1;
      bool hasErr = false;
      string errMsg = "";
      if (d < 0)
      {
         hasErr = true;
         errMsg = "Input cannot be negative";
      }
      else
      {
         try
         {
            sqRt = Math.Pow(d, 0.5);
         }
         catch (Exception e)
         {
            hasErr = true;
            errMsg = e.Message;
         }
      }
      return (sqRt, hasErr, errMsg);
   }
}
»»
□cself←'C'□cse 'Init' 'System'
```

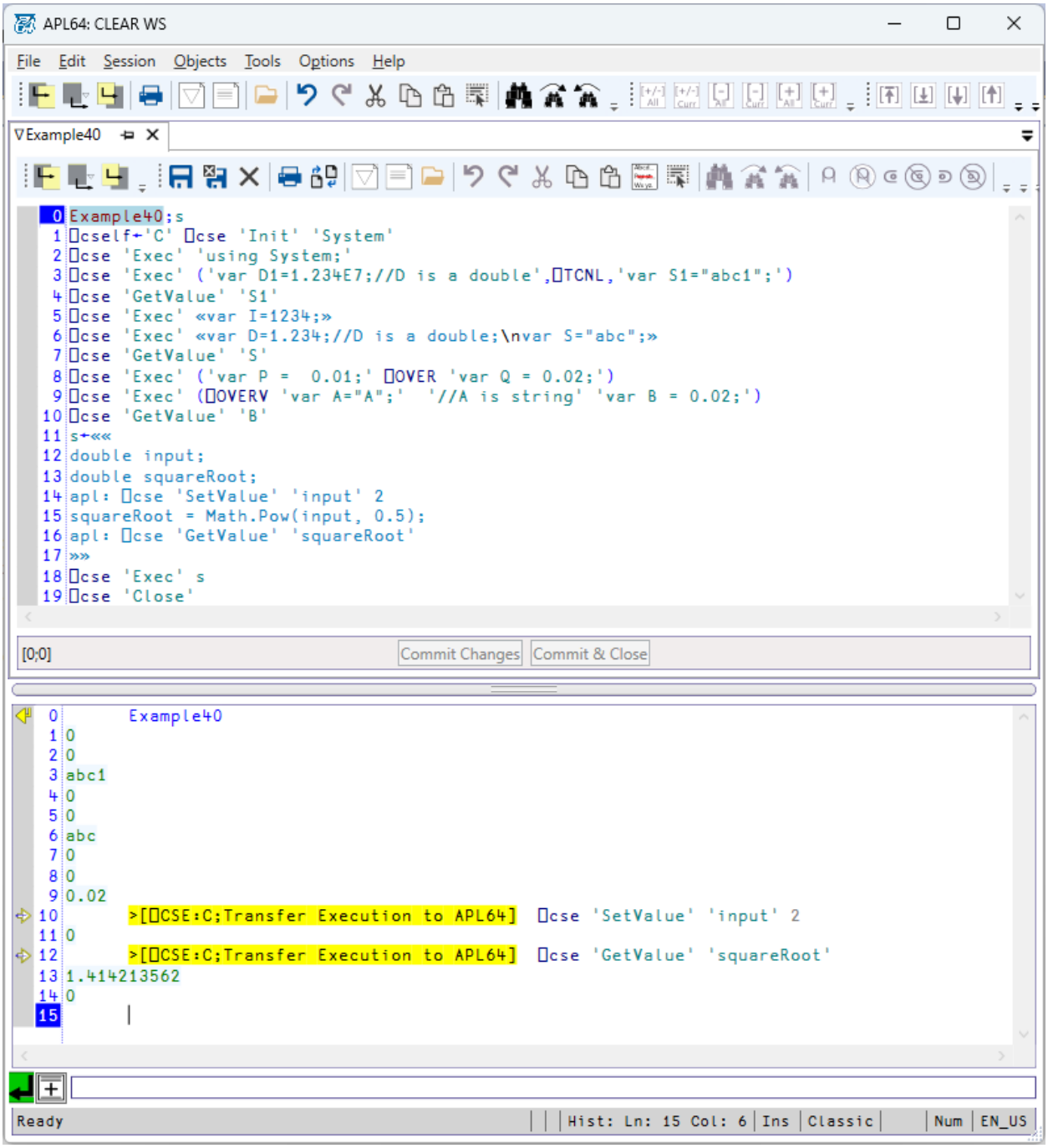
```
←□cse 'Exec' S
Z←□cse 'GetValue' 'SqRtClass.SqRt({0})' d
□cse 'Close'
```

Use the SqRt function:


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
∇SqRt
0 Z←SqRt d;S
1 S←«««
2 using System;
3 public class SqRtClass
4 {
5 public static (double sqRt, bool hasErr, string errMsg) SqRt(double d)
6 {
7 double sqRt = -1;
8 bool hasErr = false;
9 string errMsg = "";
10 if (d < 0)
11 {
12 hasErr = true;
13 errMsg = "Input cannot be negative";
14 }
15 else
16 {
17 try
18 {
19 sqRt = Math.Pow(d, 0.5);
20 }
21 catch (Exception e)
22 {
23 hasErr = true;
24 errMsg = e.Message;
25 }
26 }
27 return (sqRt, hasErr, errMsg);
28 }
29 }
30 »»»
31 ⎕cself←'C'⎕cse 'Init' 'System'
32 ←⎕cse 'Exec' S
33 Z←⎕cse 'GetValue' 'SqRtClass.SqRt({0})' d
34 ⎕cse 'Close'
[0;0]
Commit Changes
Commit & Close
0 ⎕dr¨⎕←SqRt 2
1 1.414213562 0
2 645 11 164
3 ⎕dr¨⎕←SqRt ¯2
4 ¯1 1 Input cannot be negative
5 645 11 164
6
Ready
Hist: Ln: 6 Col: 5
Ins
Classic
Num
EN_US

Example #40:
The following example illustrates single and multiple valid C# statements in the CSE script provided in the possible formats.

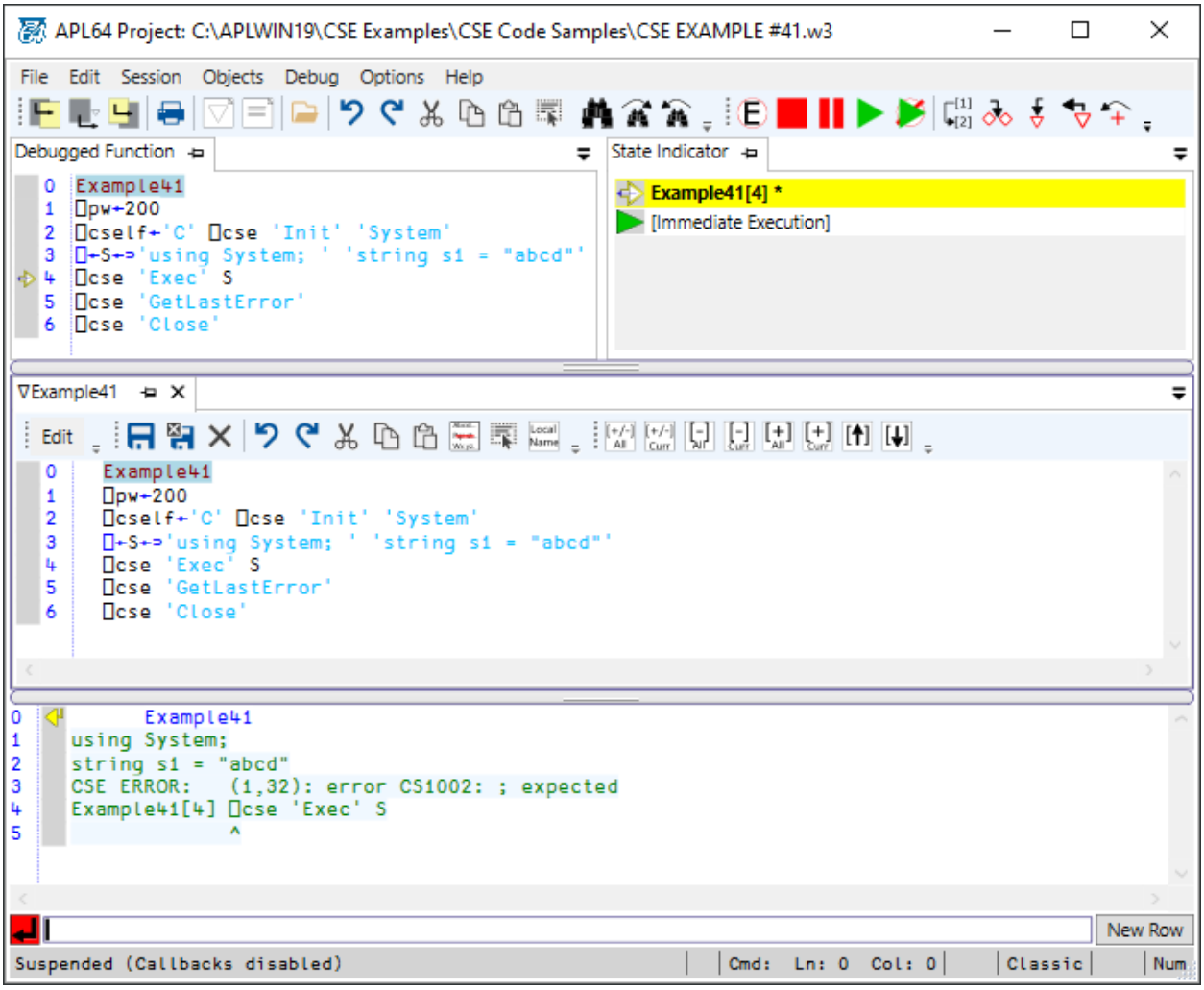
```
Example40;s
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'Exec' 'using System;'
⎕cse 'Exec' ('var D1=1.234E7;//D is a double',⎕TCNL,'var S1="abc1";')
⎕cse 'GetValue' 'S1'
⎕cse 'Exec' «var I=1234;»
⎕cse 'Exec' «var D=1.234;//D is a double;\nvar S="abc";»
⎕cse 'GetValue' 'S'
⎕cse 'Exec' ('var P =  0.01;' ⎕OVER 'var Q = 0.02;')
⎕cse 'Exec' (⎕OVERV 'var A="A";'  '//A is string' 'var B = 0.02;')
⎕cse 'GetValue' 'B'
s←««
double input;
double squareRoot;
apl: ⎕cse 'SetValue' 'input' 2
squareRoot = Math.Pow(input, 0.5);
apl: ⎕cse 'GetValue' 'squareRoot'
»»
⎕cse 'Exec' s
⎕cse 'Close'
```



Example #41:

In the following example the C# debugger identifies an invalid C# statement in the CSE script by row and column number. The Microsoft C# debugger identified the missing C# semi-colon statement separator in row #2 and column #19 of the following CSE script. Such C# errors are not 'translated' to APL64 errors so that an APL64 programmer using the CSE may directly search the C# error code and error message on Microsoft Developer Network (MSDN) to resolve the issue. The row and column number information provided in parentheses uses index origin 1.

```
Example41
⍞pw←200
⍞cself←'C' ⍞cse 'Init' 'System'
⍞←S←⊃'using System; ' 'string s1 = "abcd"'
⍞cse 'Exec' S
⍞cse 'GetLastError'
⍞cse 'Close'
```

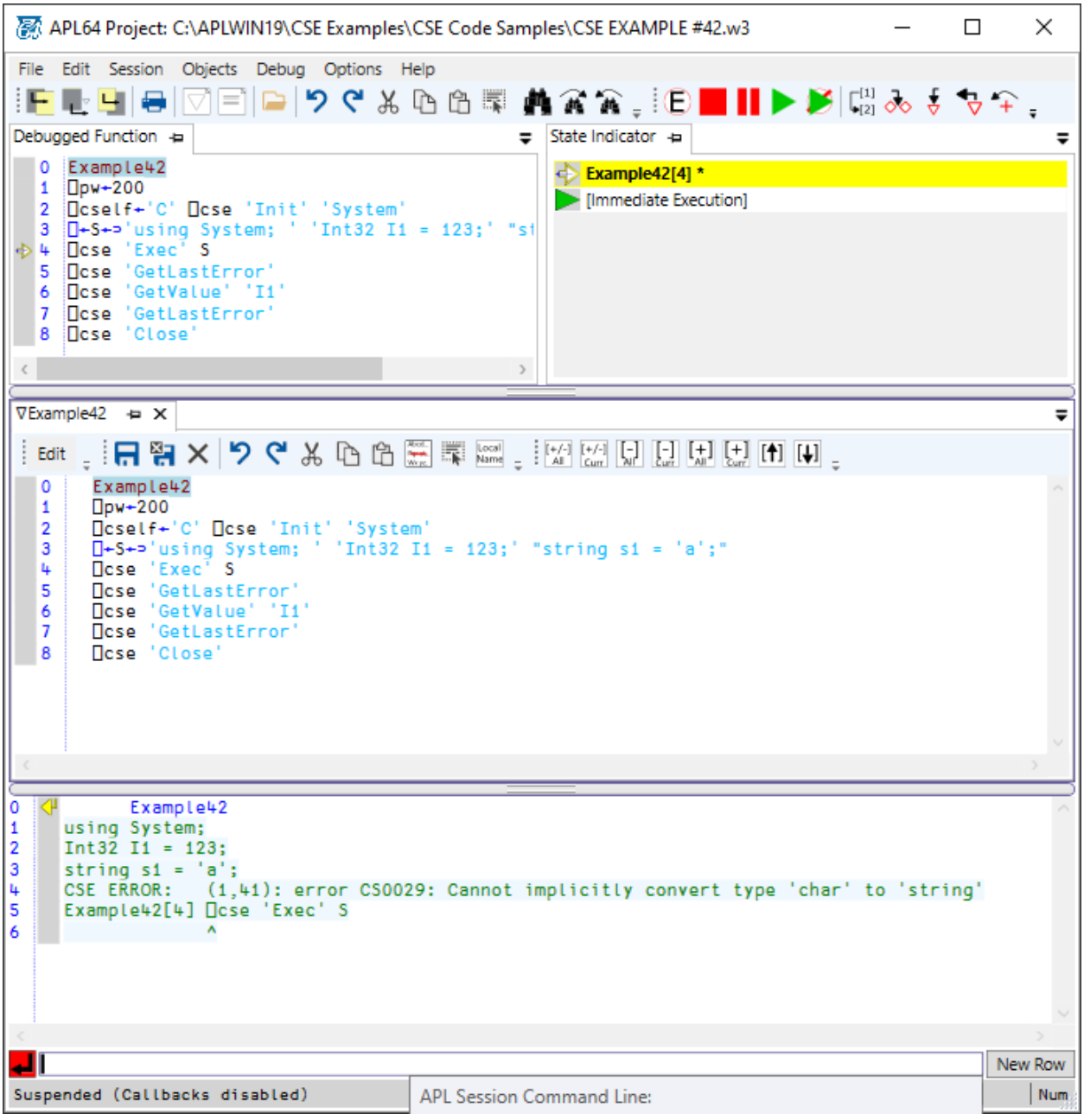


Example #42:

In the following example the C# debugger identified the script error, i.e. attempting to assign .Net character data a C# variable of type string, before actual execution occurred, so that no part of the script was successfully executed.

```
Example42
⍞pw←200
⍞cself←'C' ⍞cse 'Init' 'System'
```

```
⍞←S←⊃'using System; ' 'Int32 I1 = 123;' "string s1 = 'a';"
⍞cse 'Exec' S
⍞cse 'GetLastError'
⍞cse 'GetValue' 'I1'
⍞cse 'GetLastError'
⍞cse 'Close'
```



Example #43:

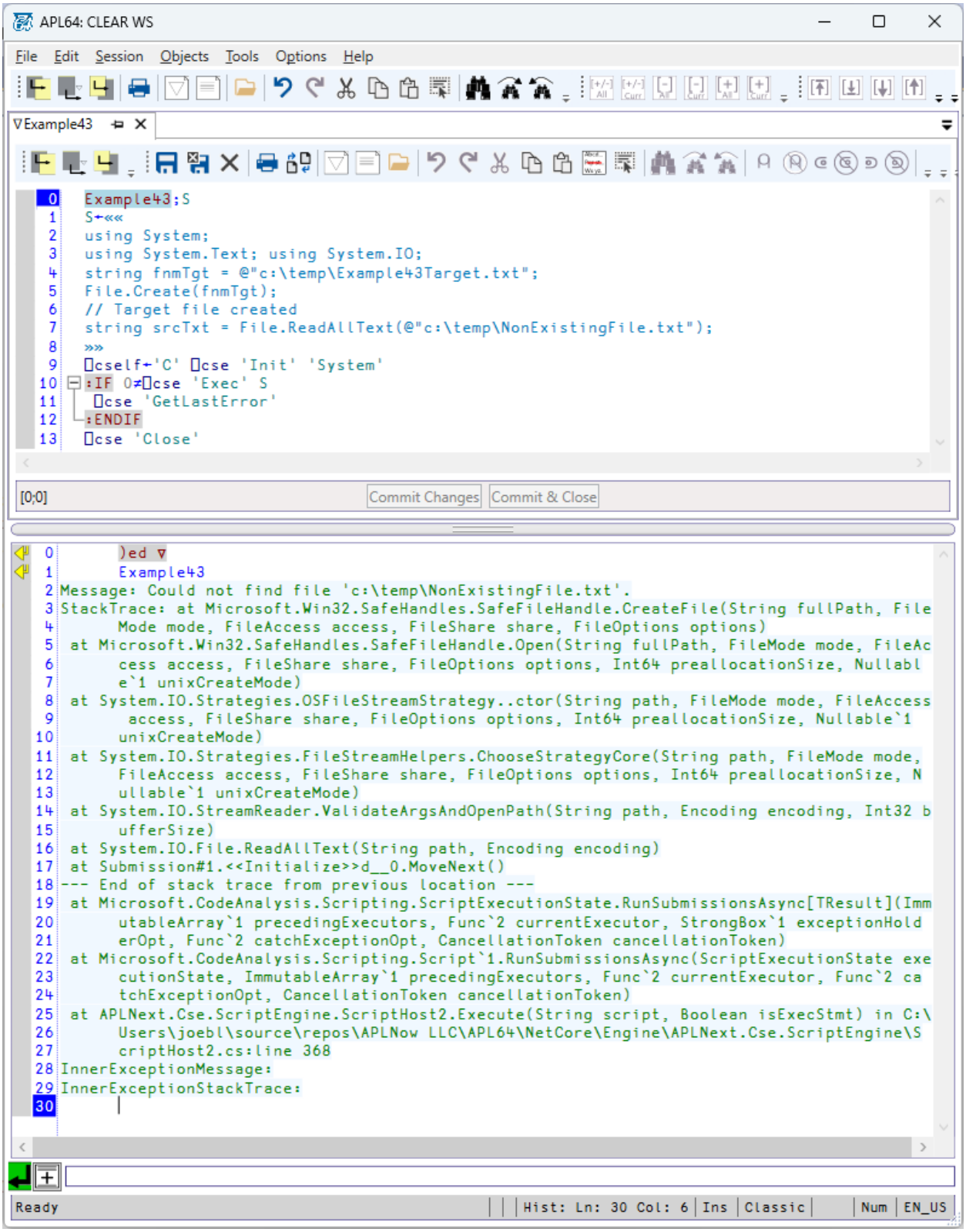
In the following example the C# debugger identified no syntax errors, so the CSE script is compiled successfully and execution was partially completed. The execution of the multi-statement C# script failed after creating a file, but before additional processing can begin. The CSE script programmer must consider the potential for an error condition which would put the instance of the CSE object in an 'intermediate' state which needs recovery.

In this case the script is a (contrived) example which could be resolved by:

- Modifying the script so that the target file is created only after the source file is known to exist or after processing of the source file is complete.
- Modifying the script to define a class which defines a C# method with try/catch/finally exception handling to identify error conditions, recover the state of the application and report the error condition.

This script did not fail the validation of the C# debugger/compiler and was compiled. The script execution was partially complete. During run time the script failed because the Example43Source.txt did not exist, so the application was put in an improper state.

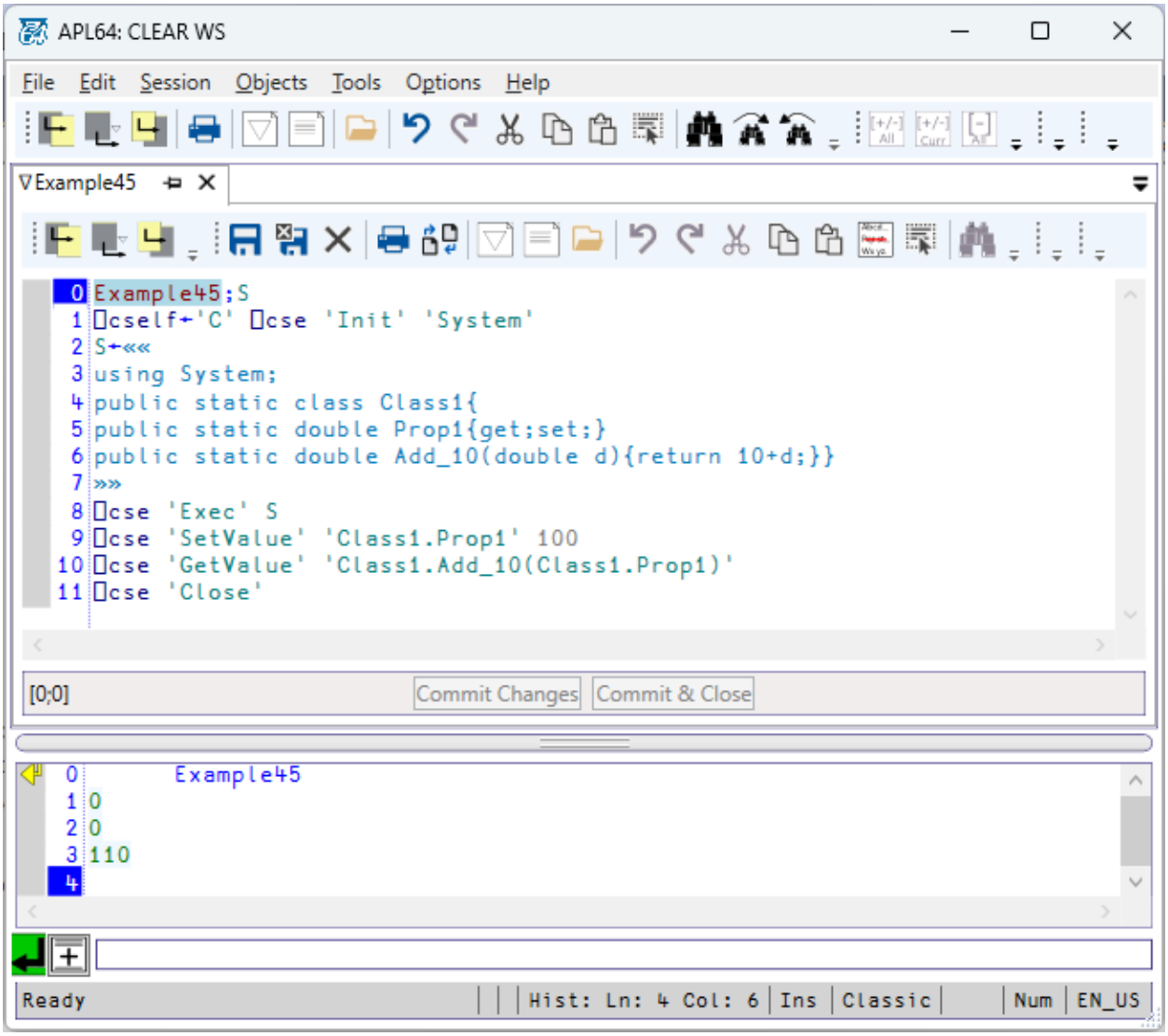
```
Example43;S
S←««
using System;
using System.Text; using System.IO;
string fnmTgt = @"c:\temp\Example43Target.txt";
File.Create(fnmTgt);
// Target file created
string srcTxt = File.ReadAllText(@"c:\temp\NonExistingFile.txt");
»»
□cself←'C' □cse 'Init' 'System'
:IF 0≠□cse 'Exec' S
 □cse 'GetLastError'
:ENDIF
□cse 'Close'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
∇Example43
Example43;S
S←⍬⍬
using System;
using System.Text; using System.IO;
string fnmTgt = @"c:\temp\Example43Target.txt";
File.Create(fnmTgt);
// Target file created
string srcTxt = File.ReadAllText(@"c:\temp\NonExistingFile.txt");
⍵⍵
⎕cself←'C' ⎕cse 'Init' 'System'
:IF 0≠⎕cse 'Exec' S
⎕cse 'GetLastError'
:ENDIF
⎕cse 'Close'
[0;0]
Commit Changes
Commit & Close
)ed ∇
Example43
Message: Could not find file 'c:\temp\NonExistingFile.txt'.
StackTrace: at Microsoft.Win32.SafeHandles.SafeFileHandle.CreateFile(String fullPath, FileMode mode, FileAccess access, FileShare share, FileOptions options)
at Microsoft.Win32.SafeHandles.SafeFileHandle.Open(String fullPath, FileMode mode, FileAccess access, FileShare share, FileOptions options, Int64 preallocationSize, Nullable`1 unixCreateMode)
at System.IO.Strategies.OSFileStreamStrategy..ctor(String path, FileMode mode, FileAccess access, FileShare share, FileOptions options, Int64 preallocationSize, Nullable`1 unixCreateMode)
at System.IO.Strategies.FileStreamHelpers.ChooseStrategyCore(String path, FileMode mode, FileAccess access, FileShare share, FileOptions options, Int64 preallocationSize, Nullable`1 unixCreateMode)
at System.IO.StreamReader.ValidateArgsAndOpenPath(String path, Encoding encoding, Int32 bufferSize)
at System.IO.File.ReadAllText(String path, Encoding encoding)
at Submission#1.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location ---
at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAsync[TResult](ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, StrongBox`1 exceptionHolderOpt, Func`2 catchExceptionOpt, CancellationToken cancellationToken)
at Microsoft.CodeAnalysis.Scripting.Script`1.RunSubmissionsAsync(ScriptExecutionState executionState, ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, Func`2 catchExceptionOpt, CancellationToken cancellationToken)
at APLNext.Cse.ScriptEngine.ScriptHost2.Execute(String script, Boolean isExecStmt) in C:\Users\joebl\source\repos\APLNow LLC\APL64\NetCore\Engine\APLNext.Cse.ScriptEngine\ScriptHost2.cs:line 368
InnerExceptionMessage:
InnerExceptionStackTrace:
Ready
Hist: Ln: 30 Col: 6
Ins
Classic
Num
EN_US

Example #45:

Using the CSE ‘Exec’ method to execute a CSE script which defines the public static class named ‘Class1’ with a public property Prop1 and a public method Add_10().

```
Example45;S
⎕cself←'C' ⎕cse 'Init' 'System'
S←««
using System;
public static class Class1{
public static double Prop1{get;set;}
public static double Add_10(double d){return 10+d;}}
»»
⎕cse 'Exec' S
⎕cse 'SetValue' 'Class1.Prop1' 100
⎕cse 'GetValue' 'Class1.Add_10(Class1.Prop1)'
⎕cse 'Close'
```

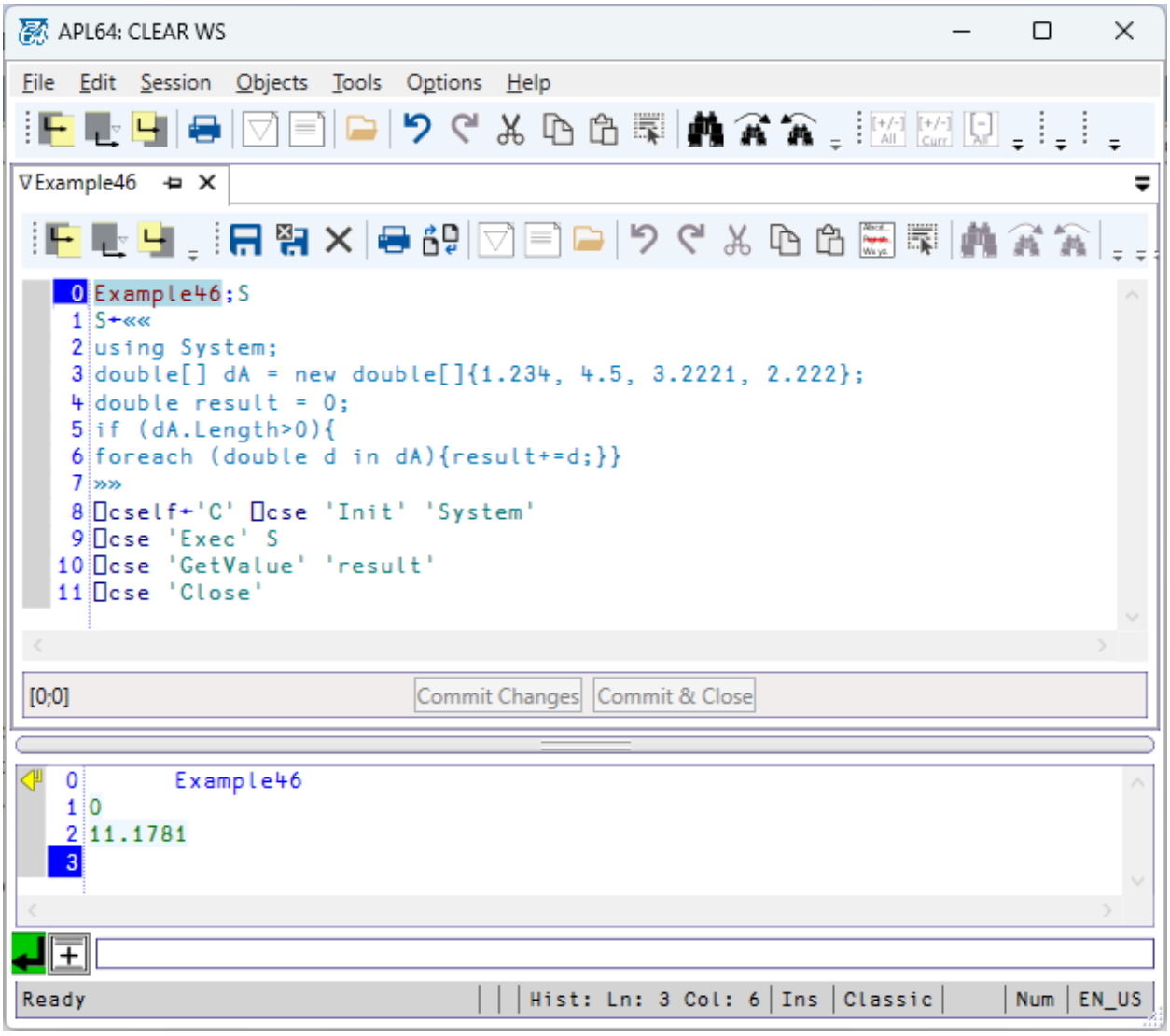


Example #46:
When C# control structures are used in a CSE script it is necessary that the entire control structure syntax be contained in one well-defined 'block' of the CSE script. For example, it is not possible to start a C# 'foreach' in one CSE script and complete the definition of such a C# control structure in a subsequently executed CSE script. The enclosing '{…}' indicates a 'block' to the C# debugger/compiler.

This example illustrates the valid use of the C# if{} and foreach(){} control structures:

```
Example46;S
S←««
using System;
double[] dA = new double[]{1.234, 4.5, 3.2221, 2.222};
double result = 0;
```

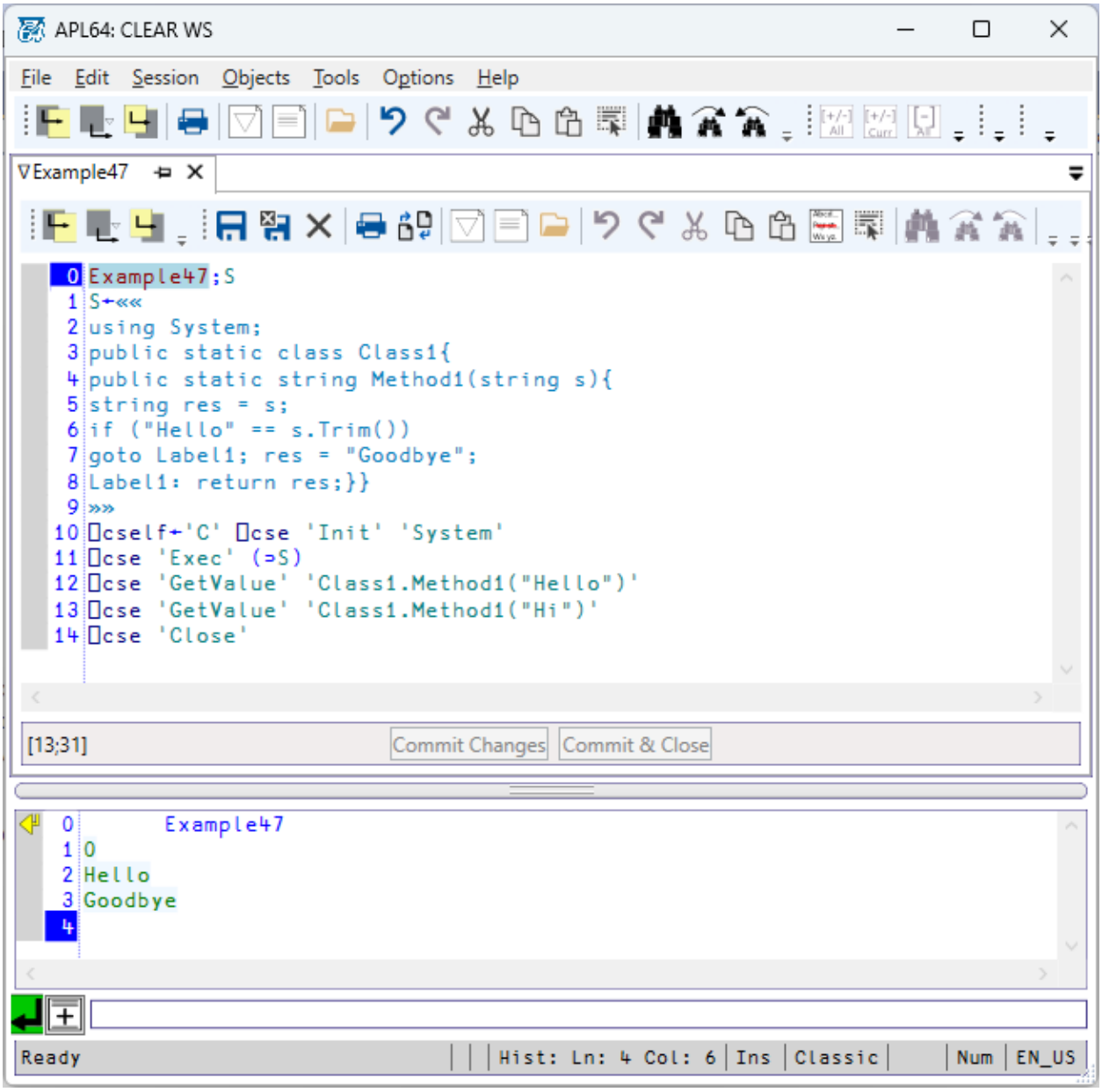
```
if (dA.Length>0){
foreach (double d in dA){result+=d;}}
»»
□cself←'C' □cse 'Init' 'System'
□cse 'Exec' S
□cse 'GetValue' 'result'
□cse 'Close'
```



Example #47:

Successful use of the C# ‘goto’ control structure requires that both the C# executable statement containing ‘goto label’ and the C# executable statement containing the ‘label’ definition be contained in the same ‘block’ of the CSE script. This example illustrates valid use of the C# ‘goto’ control structure within a C# method defined in a C# class.

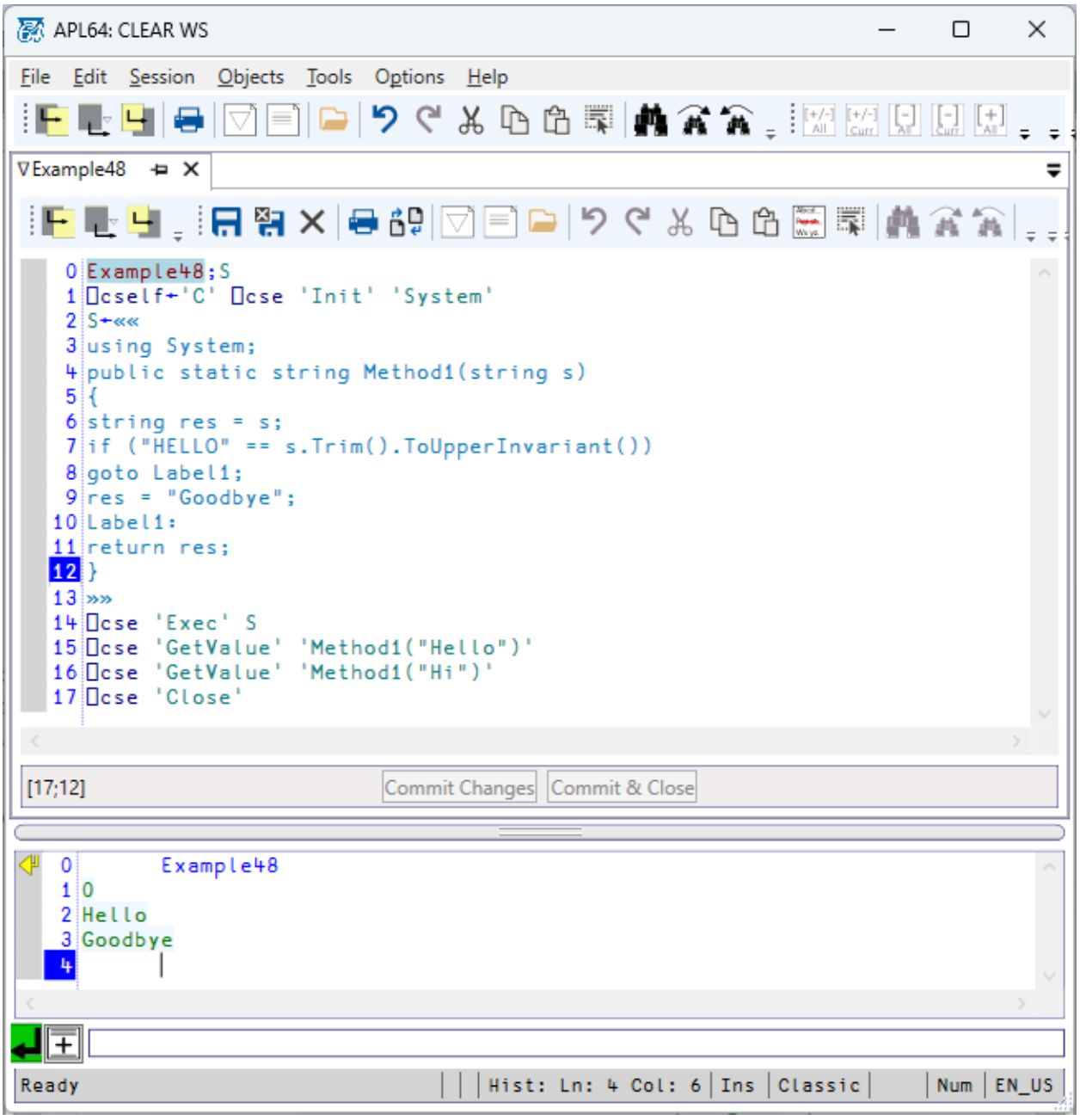
```
Example47;S
S←««
using System;
public static class Class1{
public static string Method1(string s){
string res = s;
if ("Hello" == s.Trim())
goto Label1; res = "Goodbye";
Label1: return res;}}
»»
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'Exec' S
⎕cse 'GetValue' 'Class1.Method1("Hello")'
⎕cse 'GetValue' 'Class1.Method1
```



Example #48:
Successful use of the C# ‘goto’ control structure.

```
Example48;S
□cself←'C' □cse 'Init' 'System'
S←««
using System;
public static string Method1(string s)
{
string res = s;
if ("HELLO" == s.Trim().ToUpperInvariant())
```

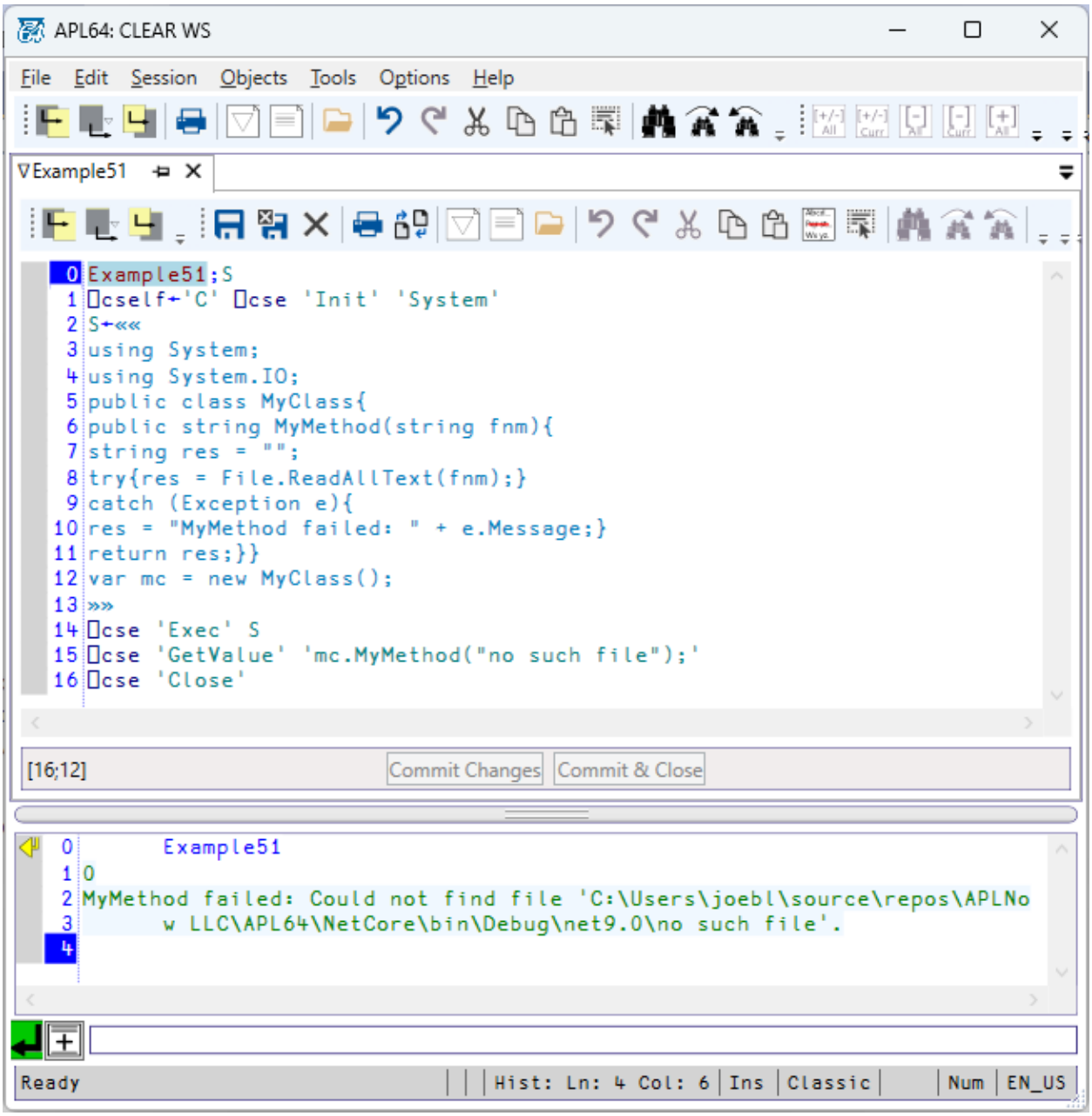
```
goto Label1;
res = "Goodbye";
Label1:
return res;
}
»»
⎕cse 'Exec' S
⎕cse 'GetValue' 'Method1("Hello")'
⎕cse 'GetValue' 'Method1("Hi")'
⎕cse 'Close'
```



Example #51:
This example illustrates the C# 'try/catch' control structure defined in a C# method.

```
Example51;S
⎕cself←'C' ⎕cse 'Init' 'System'
S←««
using System;
using System.IO;
public class MyClass{
```

```
public string MyMethod(string fnm){
string res = "";
try{res = File.ReadAllText(fnm);}
catch (Exception e){
res = "MyMethod failed: " + e.Message;}
return res;}}
var mc = new MyClass();
»»
]cse 'Exec' S
]cse 'GetValue' 'mc.MyMethod("no such file");'
]cse 'Close'
```



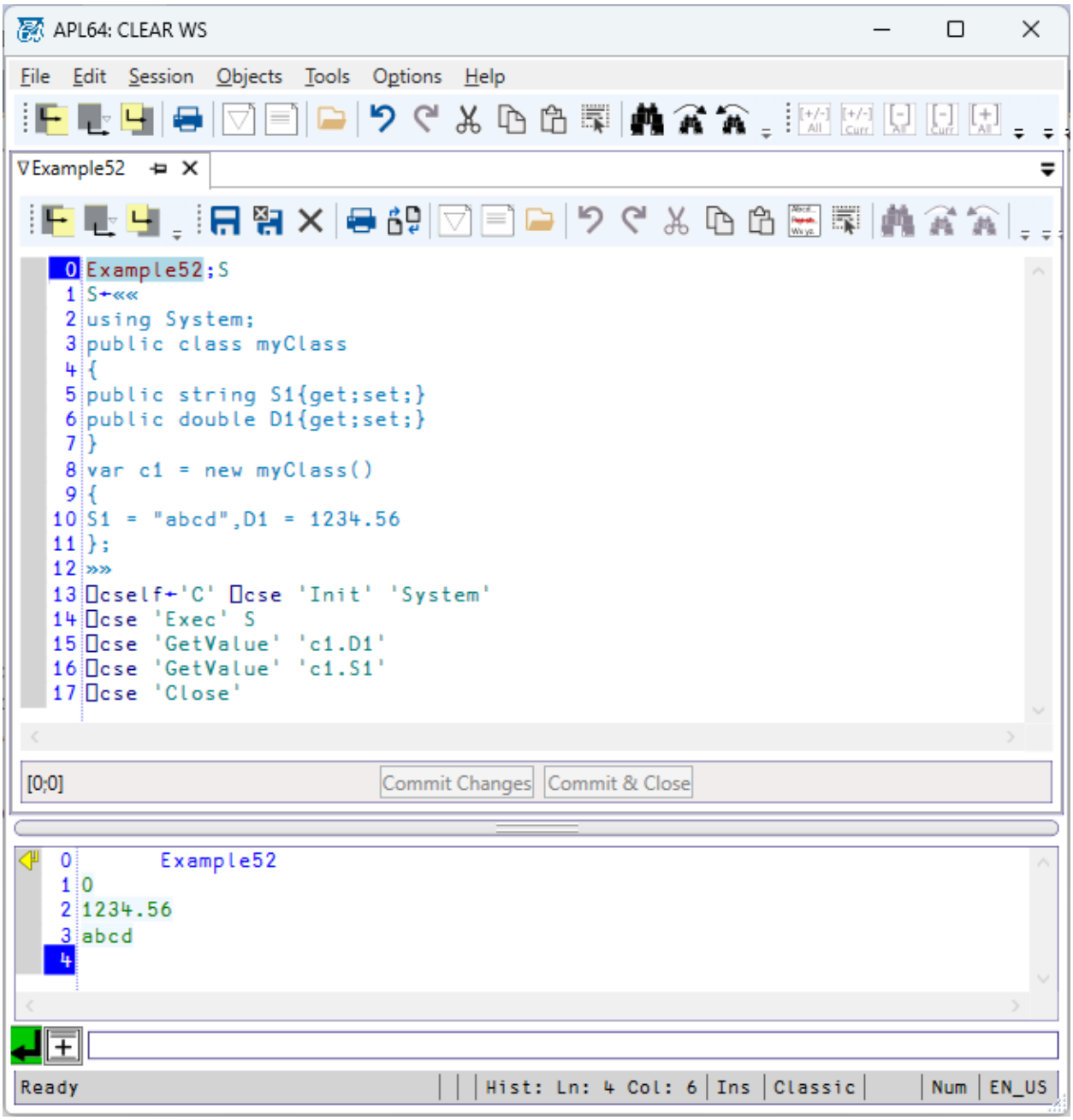
Example #52

This example illustrates a CSE script which defines a C# class, creates an instance of that class and sets the value of public properties in that instance of the class using the C# syntax:

className instanceName = new className(){propName = value,…};

```
Example52;S
S←««
using System;
public class myClass
```

```
{
public string S1{get;set;}
public double D1{get;set;}
}
var c1 = new myClass()
{
S1 = "abcd",D1 = 1234.56
};
»»
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'Exec' S
⎕cse 'GetValue' 'c1.D1'
⎕cse 'GetValue' 'c1.S1'
⎕cse 'Close'
```



CSE ExecFile Method

The operation of the CSE 'ExecFile' method is the same as that of the CSE 'Exec' method except:

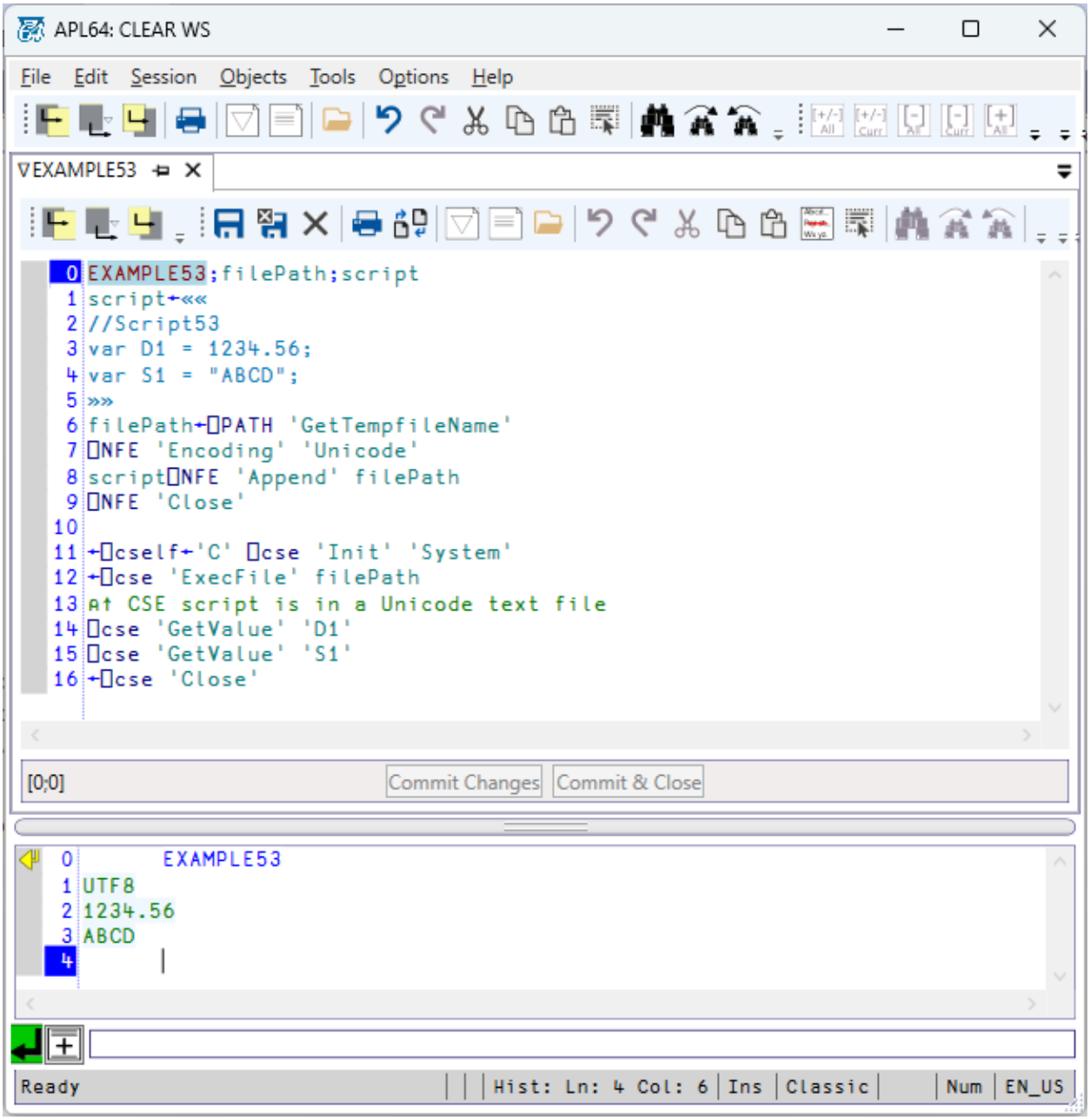
- Instead of passing the CSE script by value as the right argument of the CSE 'Exec' method, the CSE script is passed by reference to the CSE 'ExecFile' method as a fully-qualified filename.
- APL64 assumes that the CSE script file has been saved with 'ASCII', 'ANSI' or 'UTF-8 with BOM' encoding. The APL64 □N- or □XN- native file functions cannot be used to create files containing Unicode-encoded scripts, however the □NFE system `function may be used for this purpose as well as other software such as Microsoft Notepad.

- C# statements after a C# comment (ex. //comment text) in a line of the script will not be executed.

The result of the CSE ‘ExecFile’ method is affected by the value of the CSE ‘returnonerror’ property.

Example #53:
In this example a script is created in APL64 and saved as a Unicode encoded file using the the ⎕NFE system function. The script is run by the CSE ‘Exec’ method and the values of C# variables created by the script are displayed.

```
EXAMPLE53;filePath;script
script←««
//Script53
var D1 = 1234.56;
var S1 = "ABCD";
»»
filePath←⎕PATH 'GetTempfileName'
⎕NFE 'Encoding' 'Unicode'
script⎕NFE 'Append' filePath
⎕NFE 'Close'

←⎕cself←'C' ⎕cse 'Init' 'System'
←⎕cse 'ExecFile' filePath
⍝↑ CSE script is in a Unicode text file
⎕cse 'GetValue' 'D1'
⎕cse 'GetValue' 'S1'
←⎕cse 'Close'
```



Example #54

In this example a Unicode encoded script is created using Microsoft Notepad with 'UTF-8 with BOM' encoding because it contains Unicode code points outside of the range of ASCII code points. This script defines a C# method which will return a .Net string which contains elements which have Unicode code points that are not all in APL64 □AV.

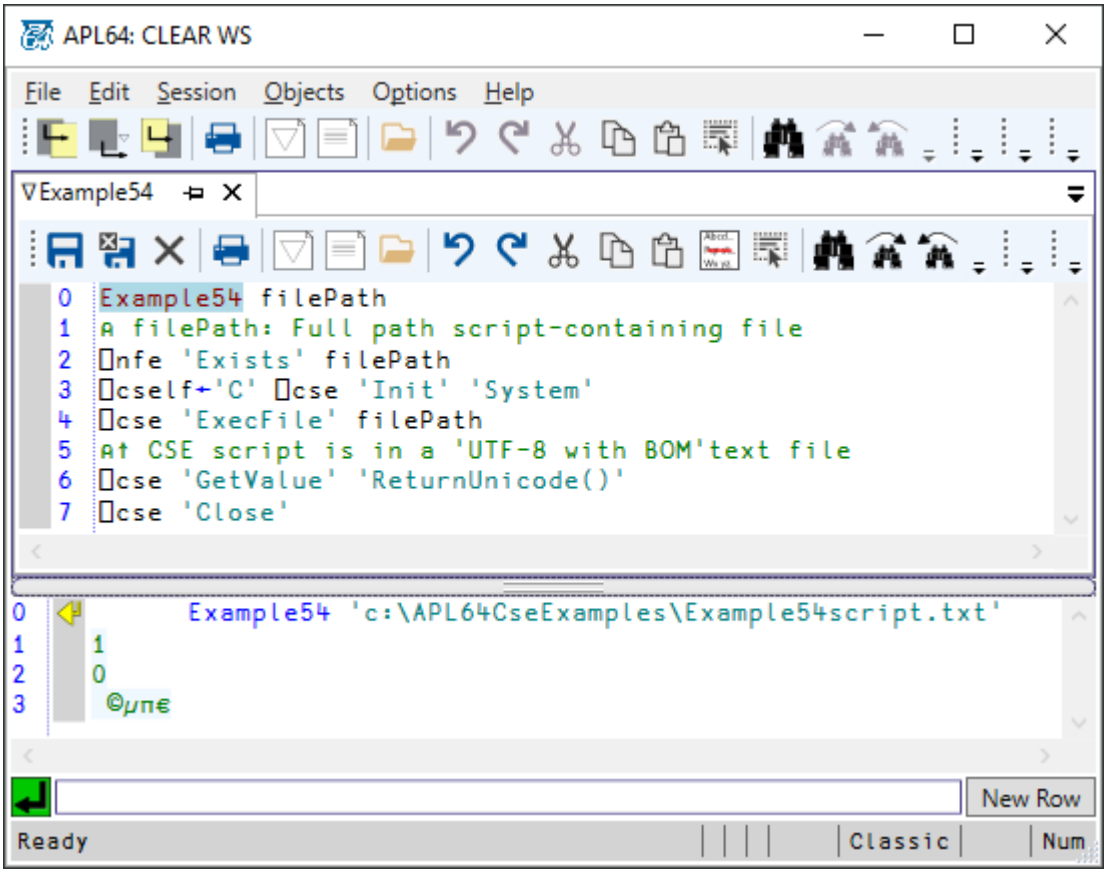
- Open Microsoft Notepad
- Copy this script and paste it into the new Notepad document:

```
public string ReturnUnicode() { return "©µπ€";}
```

- Save the Notepad file as Example54Script.txt to the desired location with the 'UTF-8 with BOM' encoding.
- Create the Example54 function in an APL64 instance using this definition:

```
Example54 filePath
⍝ filePath: Full path script-containing file
⍝nfe 'Exists' filePath
⍝cself←'C' ⍝cse 'Init' 'System'
⍝cse 'ExecFile' filePath
⍝↑ CSE script is in a 'UTF-8 with BOM'text file
⍝cse 'GetValue' 't'
⍝cse 'Close'
```

- Save the function to the current workspace
- Execute the function with the full path to the Example54Script.txt file as the right argument



CSE Script containing APL64 executable statements

In a CSE script it is possible to temporarily transfer execution control back to the APL64 instance which created the CSE instance that is executing the CSE script. This is done by including an 'APL:' CSE script statement prefix at the beginning of a row of the CSE script. Execution control will revert to the subsequent rows of a CSE script, if any, when the APL64 instance completes the execution of the APL64 statement following the 'APL:' CSE script statement prefix.

A CSE script can effectively contain executable APL64 statements in addition to executable C# statements. Only the CSE 'Exec' and 'ExecFile' methods support the 'APL:' statement prefix because only these CSE methods support multi-row CSE scripts.

A row of a CSE script containing an APL64 executable statement must:

- Include the non-case-sensitive 'APL:' prefix on the applicable row of the script such that there are non-space characters prior to the prefix on that script row.
- The APL: prefix is not case sensitive.
- Either occur at the beginning of the row of the script or be indented on that row of the script using 'space' characters.
- Not contain executable C# statements because the CSE will assume that what follows the 'APL:' prefix is an APL64 executable statement.
- Contain one or more APL64 executable statements separated by the APL64 diamond statement separator, e.g. 'apl_stmt1 ⋄ apl_stmt2'. Alternatives to using the APL64 diamond statement separator are:
    - Include multiple 'APL:'-prefixed rows in the script or
    - Put multiple APL64 executable statements into an APL64 function and put that function name in an 'APL:'-prefixed row of the script

If a script contains APL64 executable statements (using the 'APL:' statement prefix), the script must include 'new line' (e.g. ⎕TCNL) separators before and after an 'APL:'-prefixed script statement.

To run a CSE script containing an 'APL:' prefix and APL executable statements using the CSE 'ExecFile' method, the script must be encoded as Unicode.

It is possible to have more than one 'APL:' prefix on separate lines of a CSE script and hence more than one transfer of execution control to APL64 during the execution of that CSE script.

When the □cse system function determines that there is an 'embedded' APL64 executable statement on a row of a CSE script, the □cse system function will split that script into three parts:

- The initial part of the CSE script, if any, which contains no 'embedded' APL64 executable statements. If the CSE script's first row contains an 'embedded' APL64 executable statement, there is no 'initial' part.
- The row of the CSE script, if any, which contains an 'APL:' prefix followed by an 'embedded' APL64 executable statement
- The remaining rows of the CSE script, if any.

The □cse system function must split a CSE script containing an 'embedded' APL64 executable statement and execute each part separately. This is because execution control will be transferred from the .Net environment to the APL64 environment to execute the APL64 statement and then transferred back to the .Net environment to continue processing the remainder of the script.

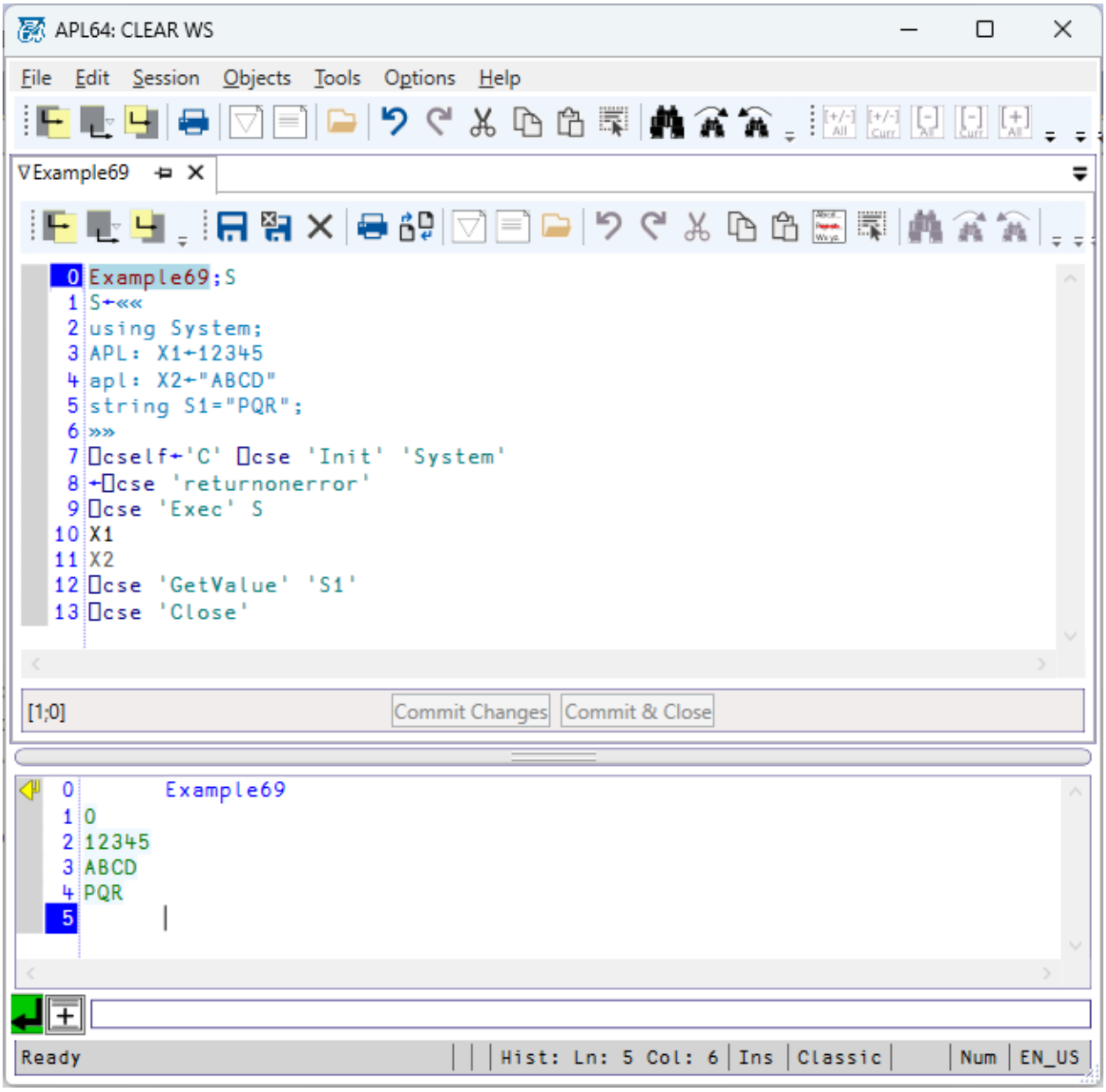
- The □cse system function will process the initial part of the CSE script, if any, assuming that it is a separate, well-defined CSE script. C# compiler error messages will be reported with reference to the row and column number of this portion of the script.
- The □cse system function will parse the row of the CSE script which contains the 'embedded' APL64 executable statement and pass that statement to the APL64 interpreter of the APL64 instance which created the CSE object executing the script.
- Finally, the □cse system function will process the remainder of the CSE script, which can potentially contain additional 'embedded' APL64 executable statements. C# compiler error messages will be reported with reference to the row and column number of this portion of the script. Such row and column numbers will not be offset by the rows or columns of a previously executed portion of the overall CSE script.

Error reporting, via □dm or the CSE 'GetLastError' method, as appropriate, will be done independently on each portion of the CSE script containing an 'embedded' APL64 executable statement.

An APL64 executable statement which uses the □cse system function or the □cself system variable can be embedded into a CSE script by using the 'APL:' prefix on the applicable CSE script row.

Example #69: Illustrates a CSE script containing several instances of 'APL:' so that execution control transferred between the script and APL64.

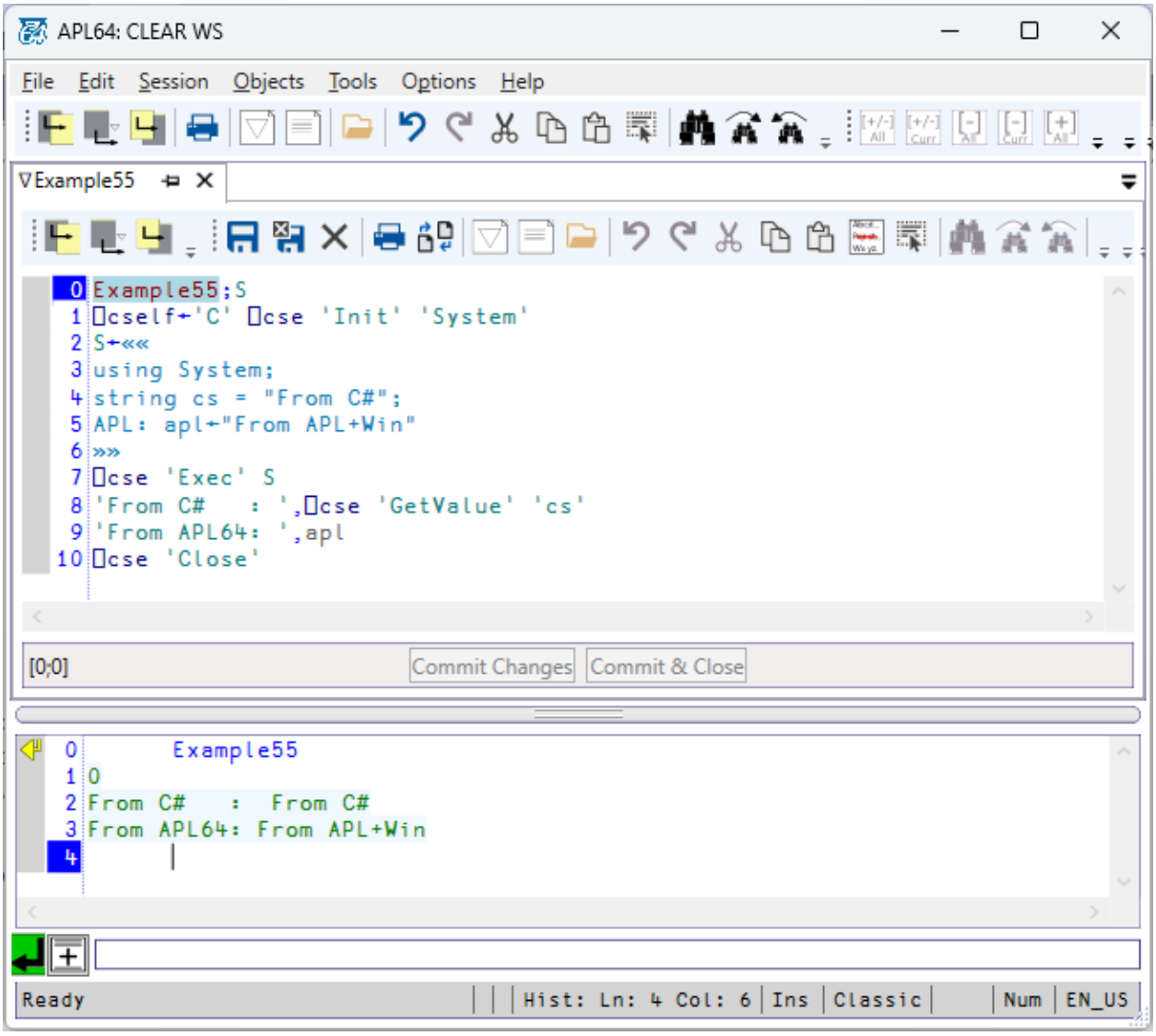
```
Example69;S
S←««
using System;
APL: X1←12345
apl: X2←"ABCD"
string S1="PQR";
»»
□cself←'C' □cse 'Init' 'System'
←□cse 'returnonerror'
□cse 'Exec' S
X1
X2
□cse 'GetValue' 'S1'
□cse 'Close'
```



Example #55:
This example illustrates a valid APL64 executable statement in a CSE script containing the ‘APL:’ prefix and run using the CSE ‘Exec’ method.

```
Example55;S
⎕cself←'C' ⎕cse 'Init' 'System'
S←««
using System;
string cs = "From C#";
APL: apl←"From APL+Win"
»»
```

```
□cse 'Exec' S
'From C#   : ',□cse 'GetValue' 'cs'
'From APL64: ',apl
□cse 'Close'
```

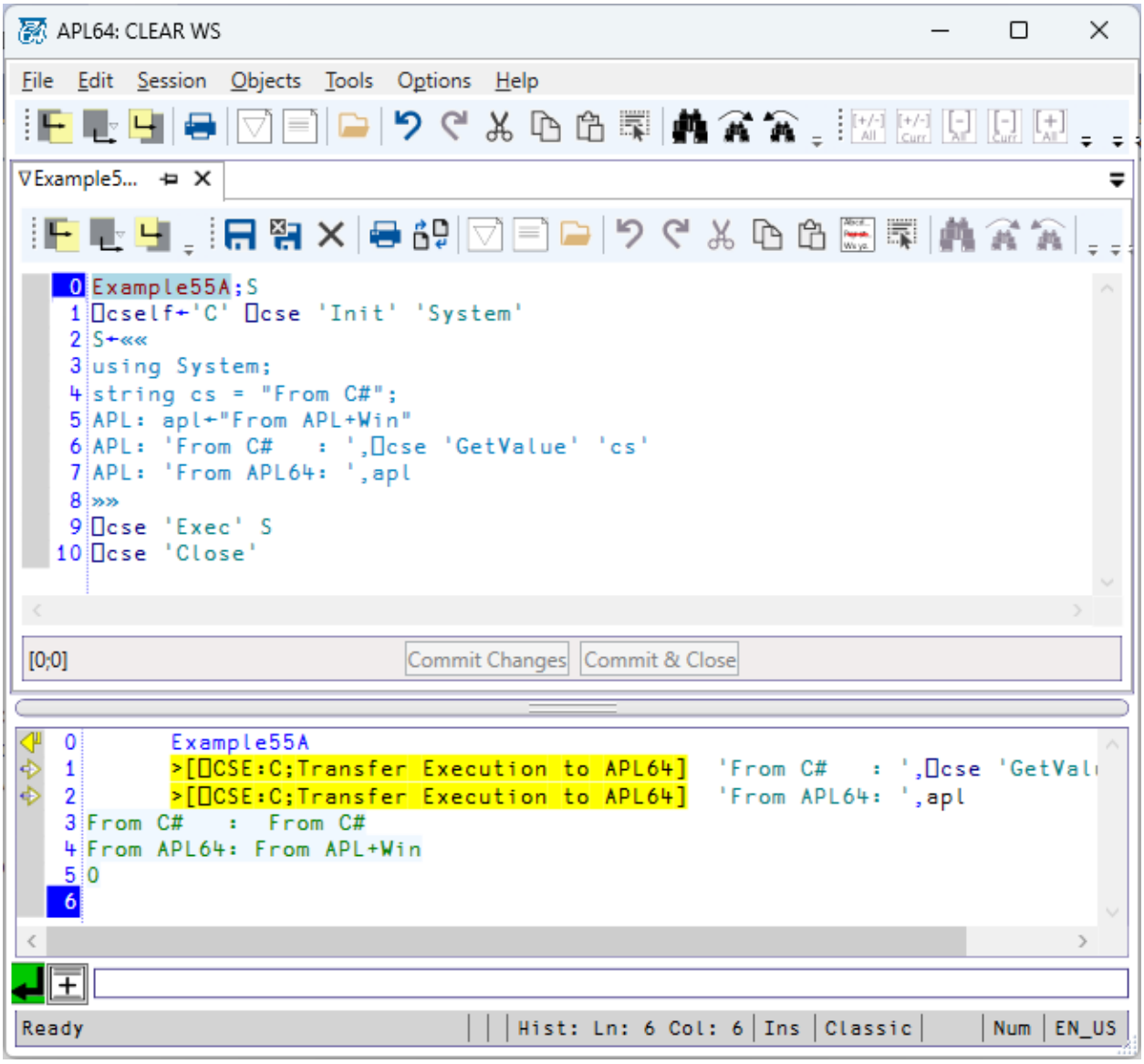


Example #55A

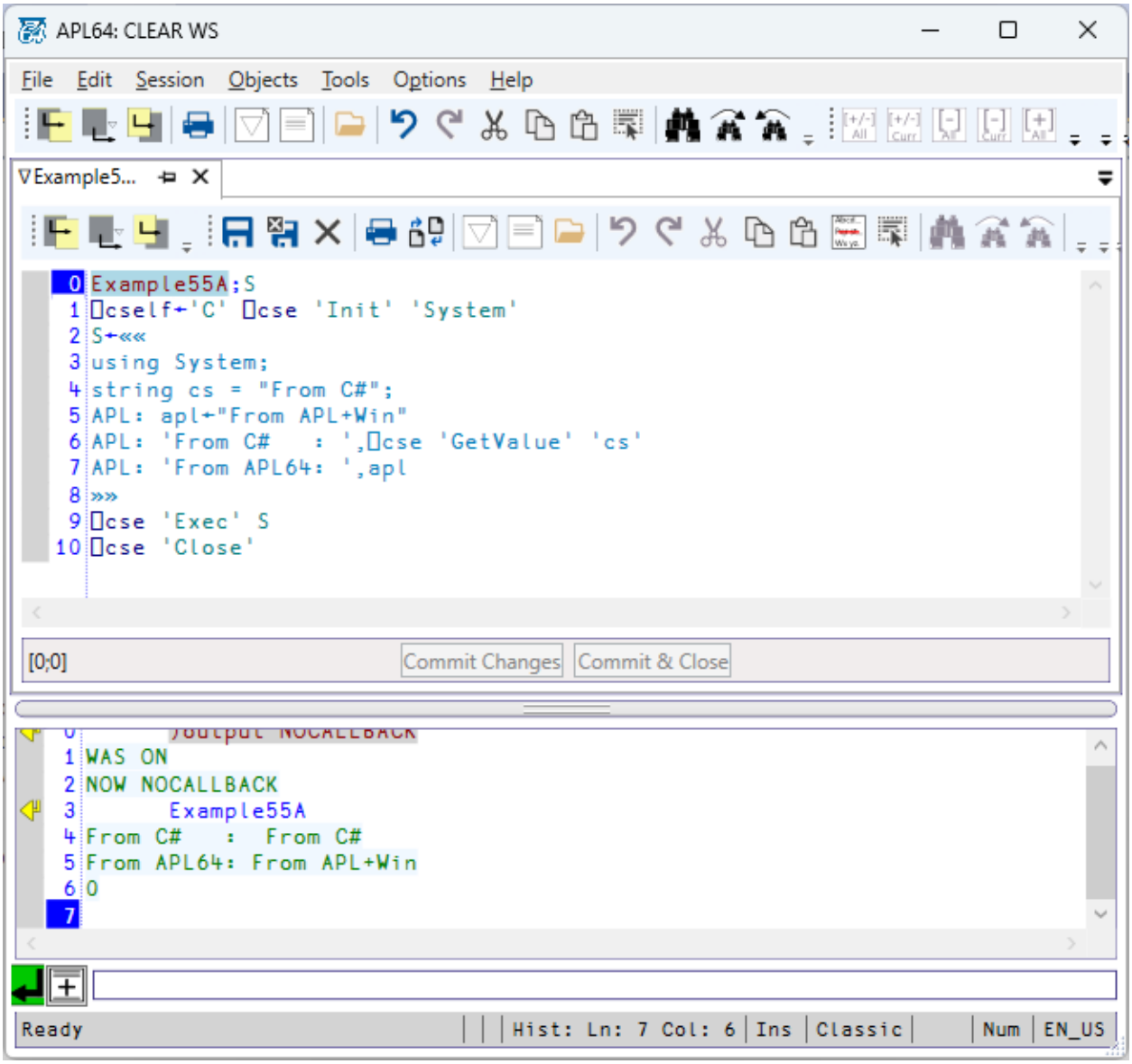
This example is the same as example 55, except that APL: prefixed statements in the script include a reference to the CSE instance itself.

```
Example55A;S
□cself←'C' □cse 'Init' 'System'
S←««
using System;
```

```
string cs = "From C#";
APL: apl←"From APL+Win"
APL: 'From C#   : ',⎕cse 'GetValue' 'cs'
APL: 'From APL64: ',apl
»»
⎕cse 'Exec' S
⎕cse 'Close'
```



If desired, use the APL64 system command)output NOCALLBACK to suppress the call back information in the history pane:

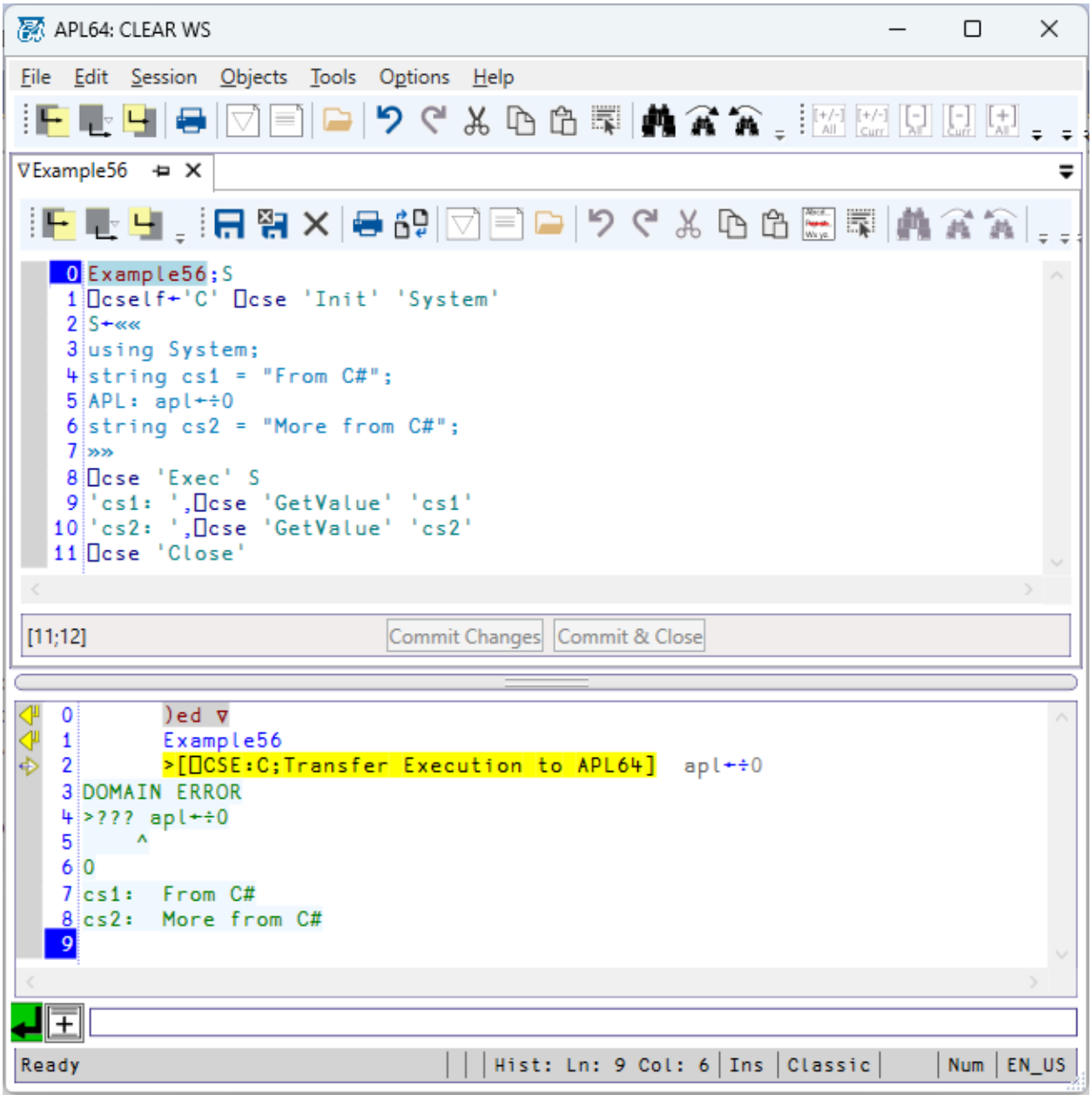


Example #56:

This example illustrates how an APL64 error in an APL64 executable statement embedded in a CSE script is reported by ⎕dm. The first part of the script, creating the C# variable 'cs1' is successfully executed. Because of the error in the embedded APL64 executable statement in the second part of the script, the third part of the script, creating the C# variable 'cs1' was not executed.

```
Example56;S
⎕cself←'C' ⎕cse 'Init' 'System'
S←««
using System;
string cs1 = "From C#";
APL: apl←÷0
```

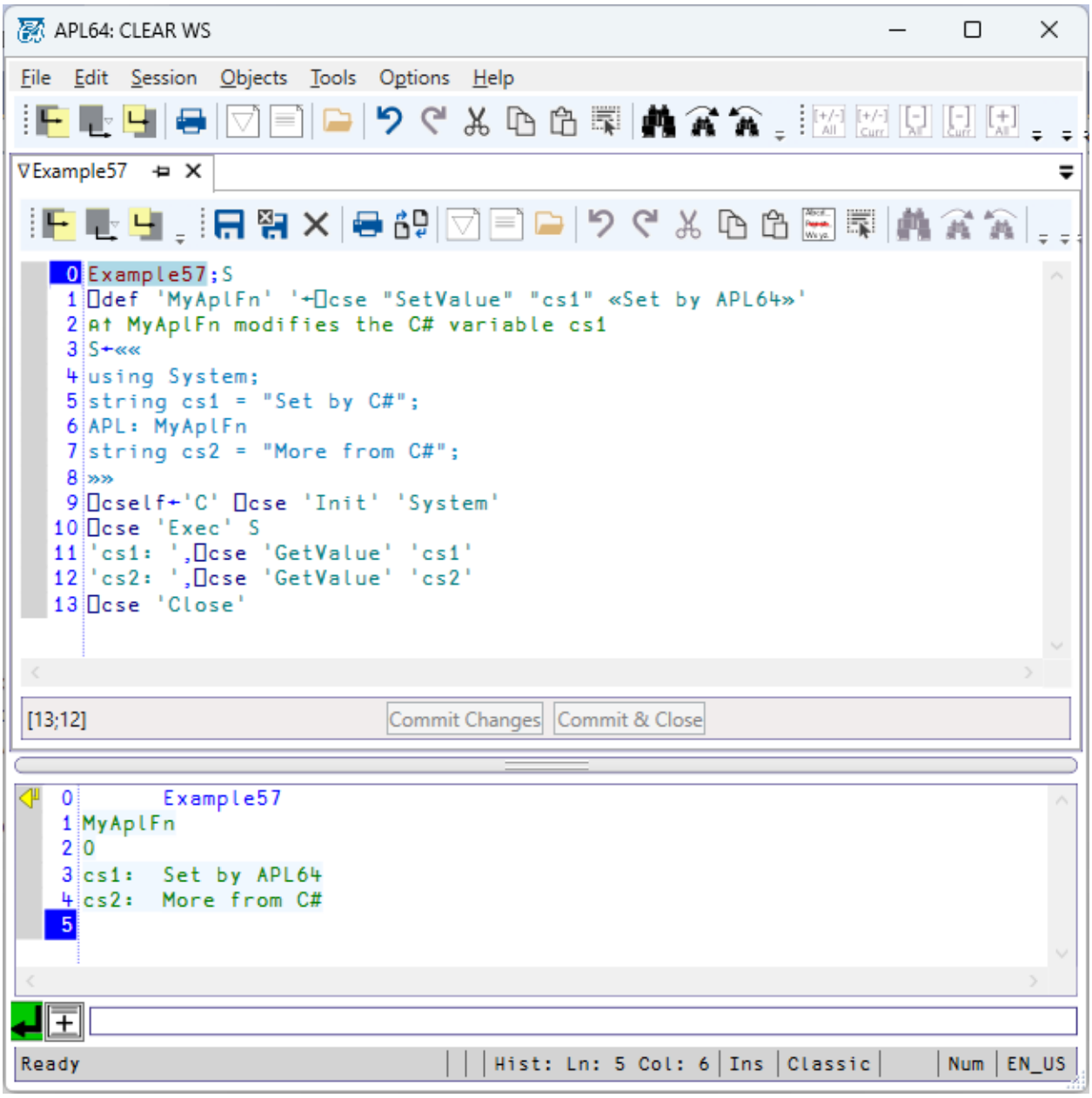
```
string cs2 = "More from C#";
»»
□cse 'Exec' S
'cs1: ',□cse 'GetValue' 'cs1'
'cs2: ',□cse 'GetValue' 'cs2'
□cse 'Close'
```



Example #57:

CSE script with an embedded APL64 executable statement which runs the 'MyAplFn' function which modifies the C# object 'cs1' created by the CSE script.

```
Example57;S
⎕def 'MyAplFn' '←⎕cse "SetValue" "cs1" «Set by APL64»'
⍝↑ MyAplFn modifies the C# variable cs1
S←««
using System;
string cs1 = "Set by C#";
APL: MyAplFn
string cs2 = "More from C#";
»»
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'Exec' S
'cs1: ',⎕cse 'GetValue' 'cs1'
'cs2: ',⎕cse 'GetValue' 'cs2'
⎕cse 'Close'
```

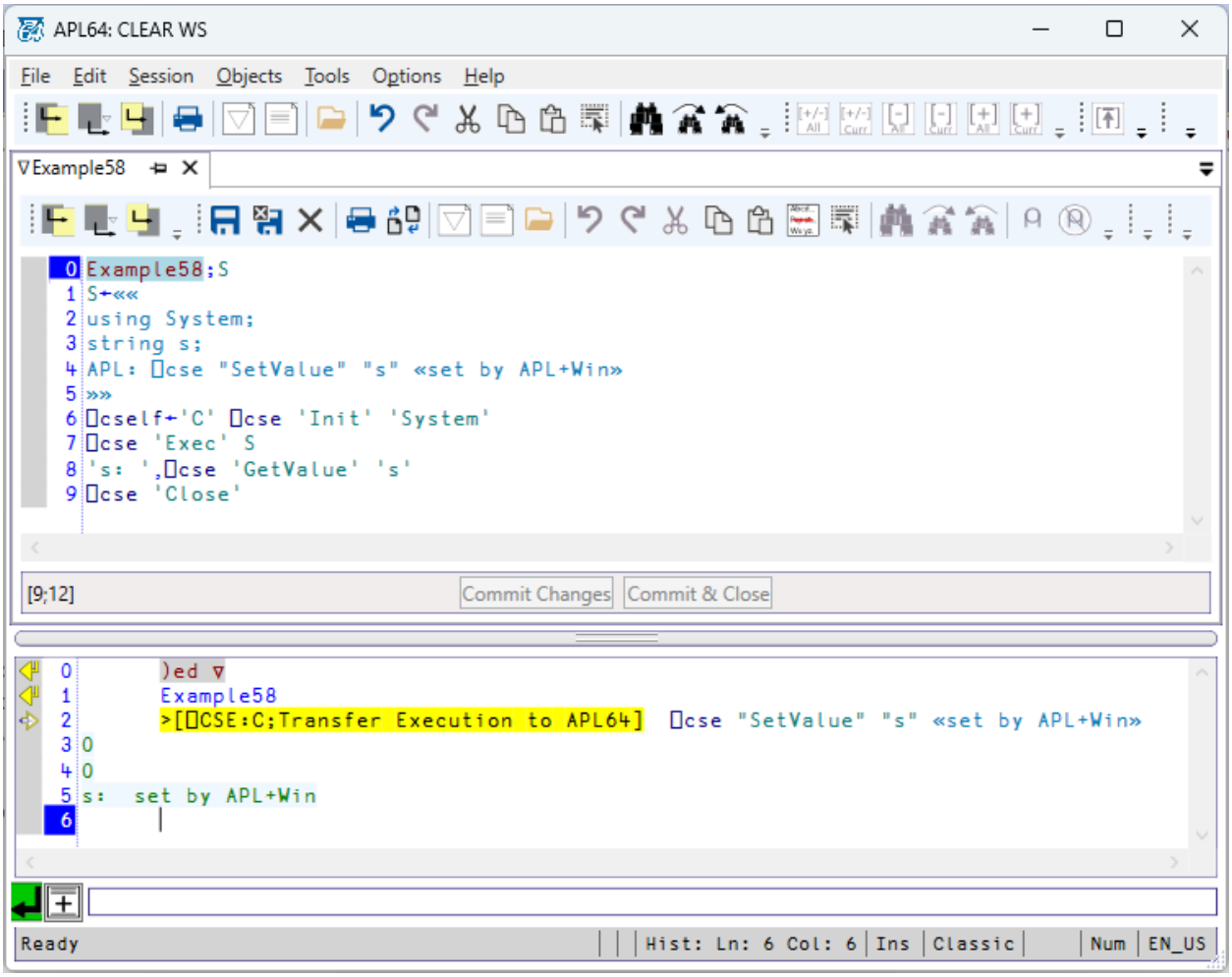


Example #58

CSE script with an embedded APL64 executable statement which uses the □cse system function within the script to successfully manipulate the CSE object executing the script.

```
Example58;S
S←««
using System;
string s;
APL: ⎕cse "SetValue" "s" «set by APL+Win»
»»
⎕cself←'C' ⎕cse 'Init' 'System'
```

```
□cse 'Exec' S
's: ',□cse 'GetValue' 's'
□cse 'Close'
```

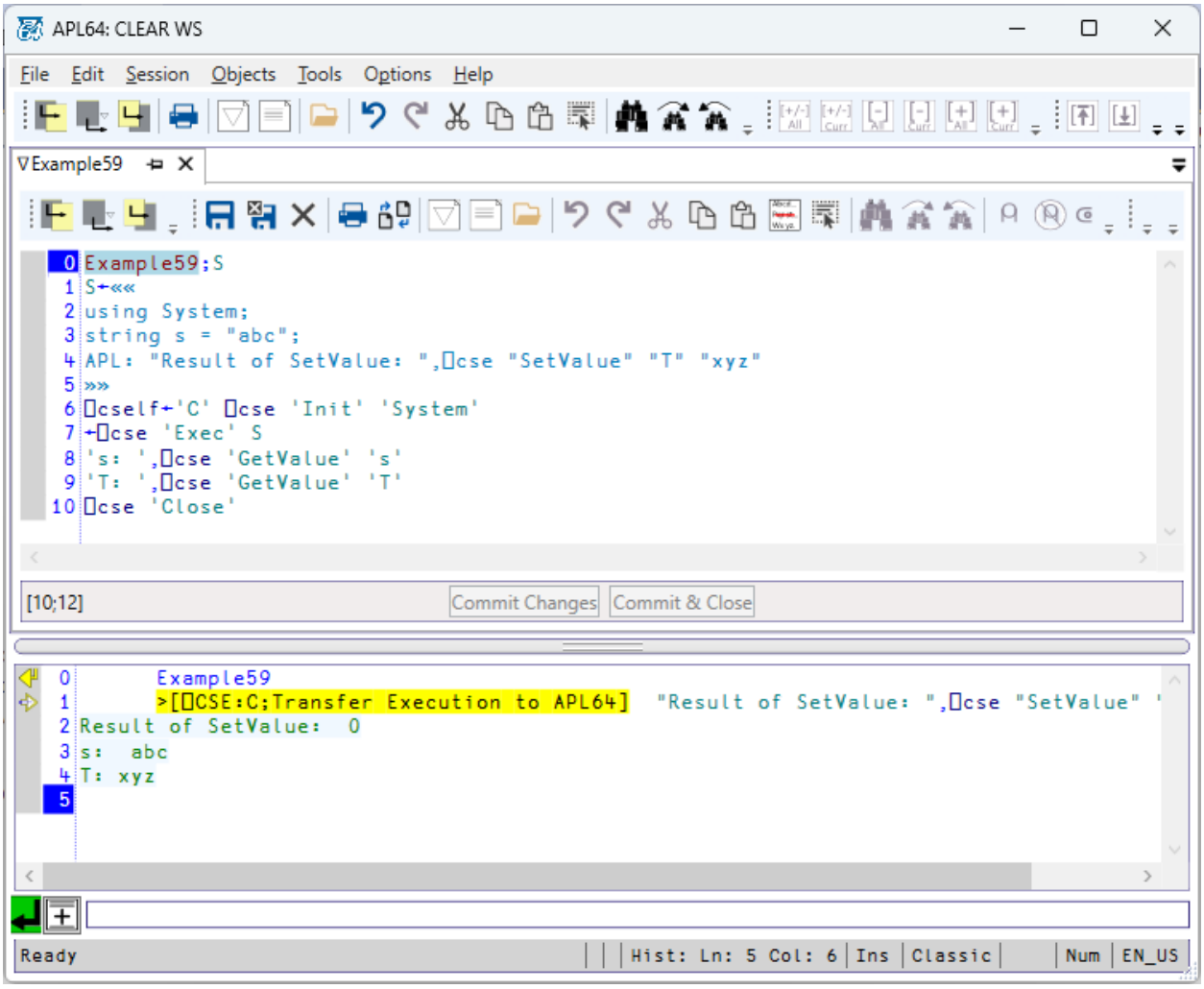


Example #59

CSE script containing an APL64 executable statement. When that APL64 executable statement gains execution control, it uses the CSE to modify the current CSE instance.

```
Example59;S
S←««
using System;
string s = "abc";
APL: "Result of SetValue: ",□cse "SetValue" "T" "xyz"
»»
□cself←'C' □cse 'Init' 'System'
←□cse 'Exec' S
's: ',□cse 'GetValue' 's'
```

```
'T: ',□cse 'GetValue' 'T'
□cse 'Close'
```

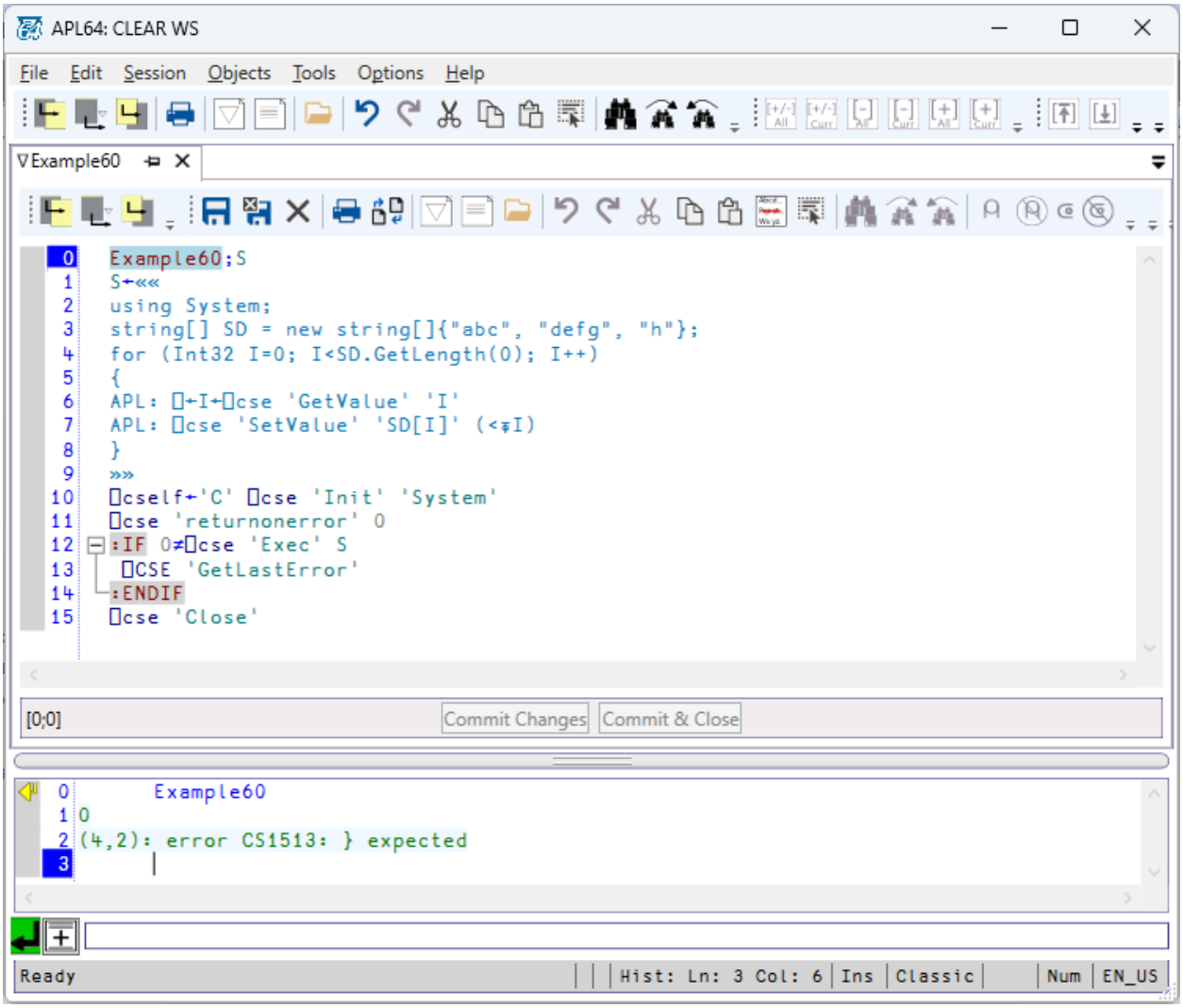


Example #60

This example illustrates that it is not possible to embed APL64 executable statements within a C# control structure in a CSE script. It is necessary for each portion of a script, before and after an embedded APL64 executable statement, to be a separate and well-defined CSE script with complete C# control structures. Because execution control is transferred from .Net to APL64 and back when there is an embedded APL64 executable statement in a CSE script, the □cse system function splits the script execution to the portions before and after the embedded APL64 executable statement.

When the script is executed using the CSE 'Exec' method, the C# compiler error indicates that the first part of the script (before the APL: I←C □cse 'GetValue' 'I' script row) does not contain a well-defined, i.e. complete, C# 'for (){}' control structure.

```
Example60;S
S←««
using System;
string[] SD = new string[]{"abc", "defg", "h"};
for (Int32 I=0; I<SD.GetLength(0); I++)
{
APL: □←I←□cse 'GetValue' 'I'
APL: □cse 'SetValue' 'SD[I]' (<⍤⊖ I)
}
»»
□cself←'C' □cse 'Init' 'System'
□cse 'returnonerror' 0
:IF 0≠□cse 'Exec' S
 □CSE 'GetLastError'
:ENDIF
□cse 'Close'
```

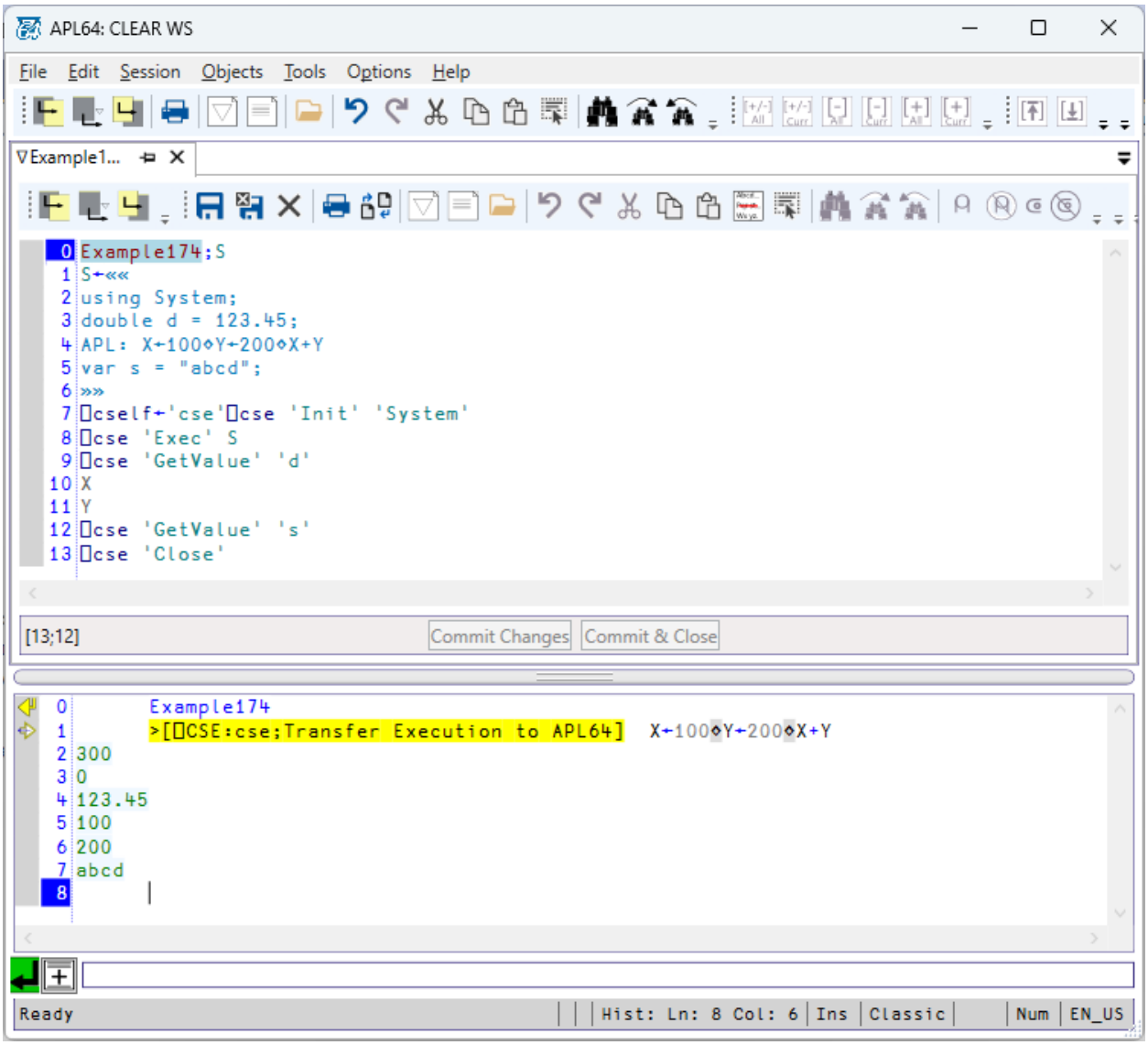


Example #174

This example illustrates multiple APL64 executable statements separated by the APL64 diamond statement separator on a row of a CSE script prefixed by 'APL:'.

```
Example174;S
S←««
using System;
double d = 123.45;
APL: X←100⋄Y←200⋄X+Y
var s = "abcd";
»»
□cself←'cse'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'GetValue' 'd'
X
```

```
Y
□cse 'GetValue' 's'
□cse 'Close'
```

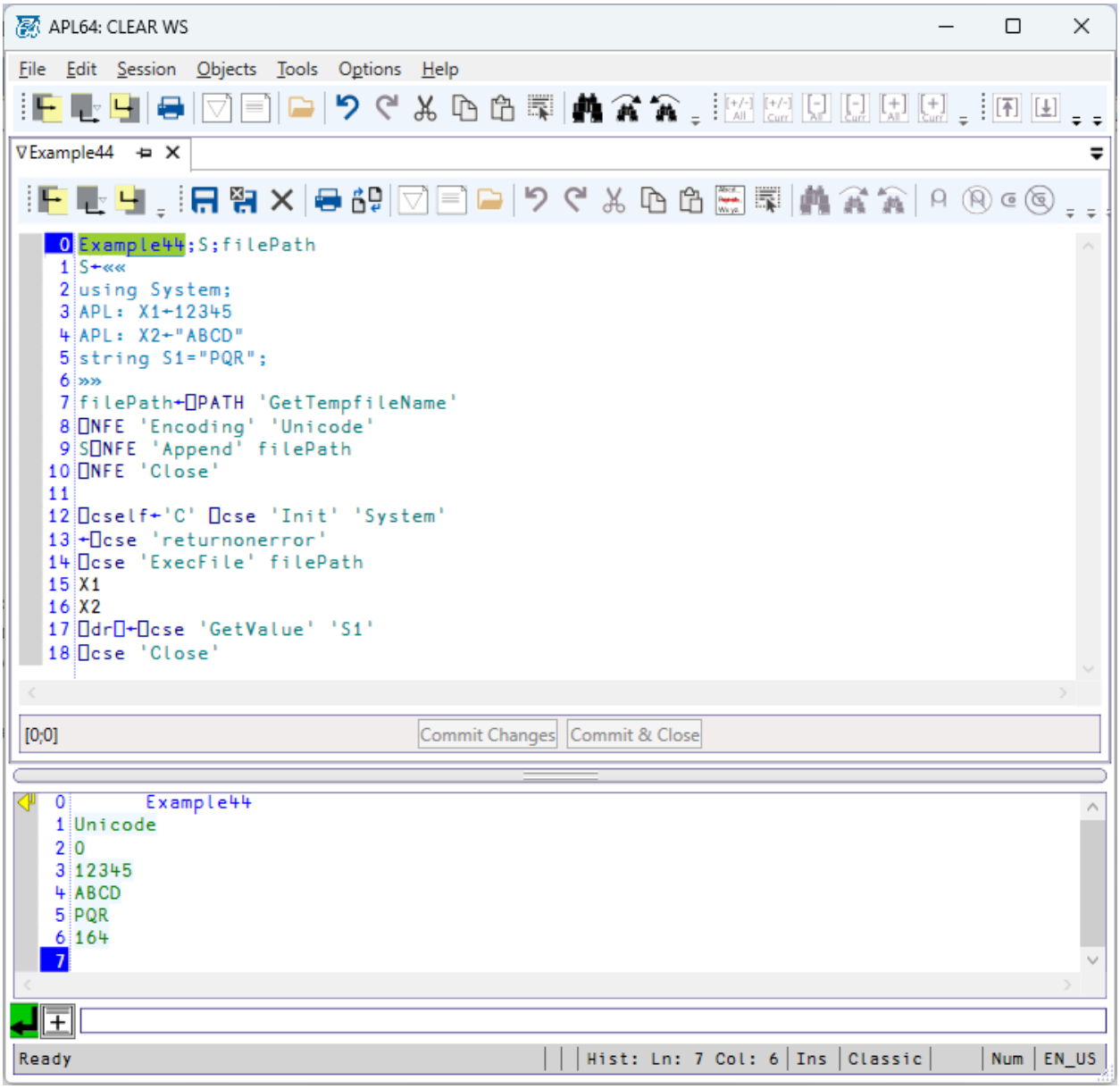


Example #44

In this example a filed C# script containing APL64 executable statements prefixed by ‘APL:’ is illustrated.

```
Example44;S;filePath
S←««
using System;
APL: X1←12345
APL: X2←"ABCD"
string S1="PQR";
```

```
»»
filePath←⎕PATH 'GetTempfileName'
⎕NFE 'Encoding' 'Unicode'
S⎕NFE 'Append' filePath
⎕NFE 'Close'

⎕cself←'C' ⎕cse 'Init' 'System'
←⎕cse 'returnonerror'
⎕cse 'ExecFile' filePath
X1
X2
⎕dr⎕←⎕cse 'GetValue' 'S1'
⎕cse 'Close'
```



CSE ExecStmt Method

The 'ExecStmt' method is the same as the CSE Exec method except:

- The right argument must be an APL64 character vector or scalar string
- The 'ExecStmt' method is not designed to support the 'APL:' execution control transfer option
- The 'ExecStmt' method supports value substitution of APL64 values
- C# statements after a C# comment (ex. //comment text) in the right argument will not be executed only if a new line character (⎕tcnl) after the comment separates that comment from subsequent C# statements in the C# script.

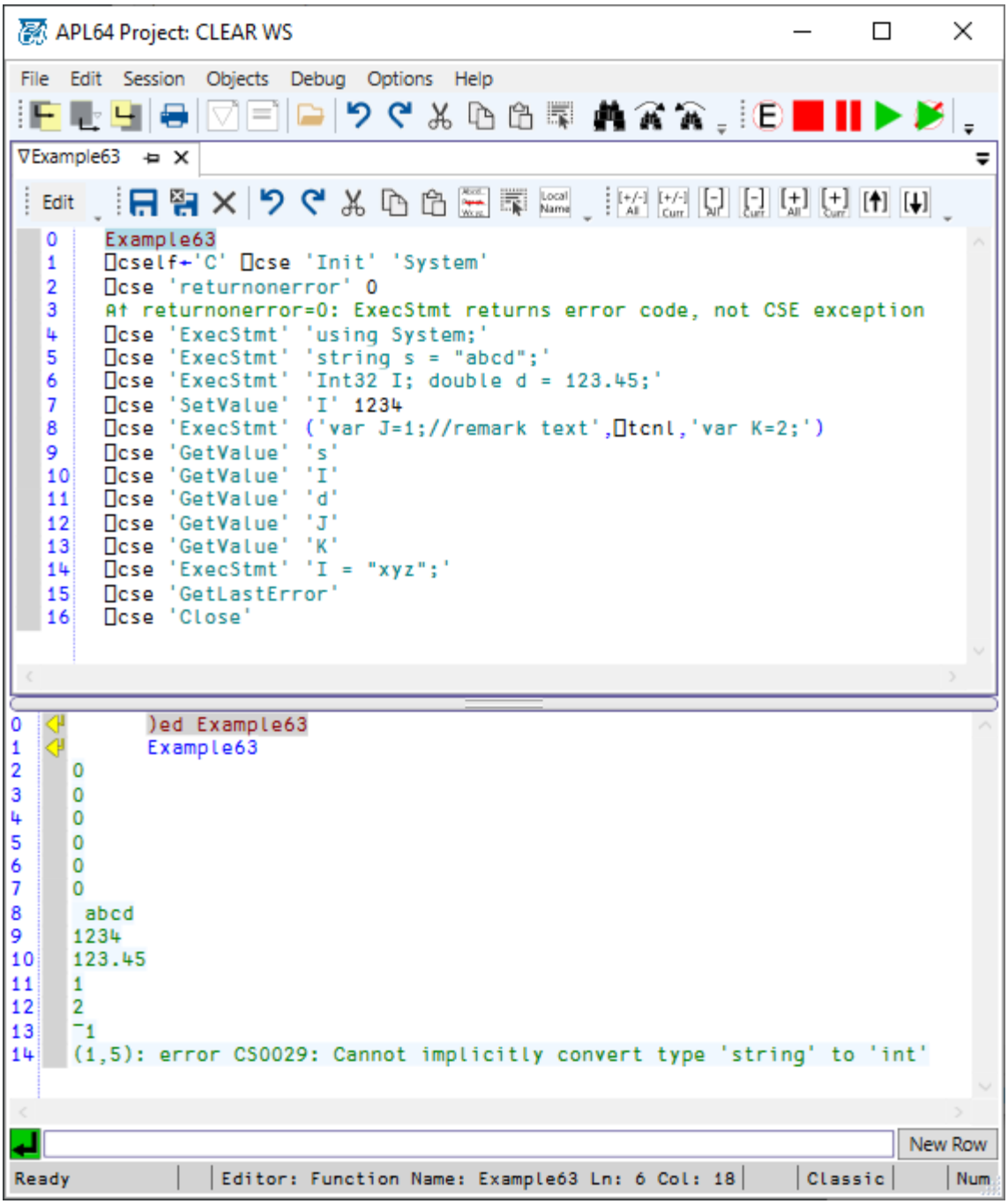
The result of the CSE 'ExecStmt' method is affected by the value of the CSE 'returnonerror' property.

The right argument of the CSE ‘ExecStmt’ method can contain one or a collection of valid C# statements. The validity of the C# statement is verified using the Microsoft C# debugger within the existing state of the instance of the CSE object. Only C# executable statements may be included in the right argument of the CSE ‘ExecStmt’ method. The right argument of the CSE ‘ExecStmt’ methods cannot contain the ‘APL:’ execution control transfer prefix.

When a 1-line CSE script is to be executed, the CSE ‘ExecStmt’ may provide a more convenient syntax compared to the syntax of the CSE ‘Exec’ or ‘ExecFile’ methods.

Example #63:
Several CSE ‘ExecStmt’ examples are illustrated. Because the value of the CSE ‘returnonerror’ property value is zero, the result of the CSE ‘ExecStmt’ method is: 0/Success -1/C# exception.

```
Example63
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
⍝↑ returnonerror=0: ExecStmt returns error code, not CSE exception
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'string s = "abcd";'
⎕cse 'ExecStmt' 'Int32 I; double d = 123.45;'
⎕cse 'SetValue' 'I' 1234
⎕cse 'ExecStmt' ('var J=1;//remark text',⎕tcnl,'var K=2;')
⎕cse 'GetValue' 's'
⎕cse 'GetValue' 'I'
⎕cse 'GetValue' 'd'
⎕cse 'GetValue' 'J'
⎕cse 'GetValue' 'K'
⎕cse 'ExecStmt' 'I = "xyz";'
⎕cse 'GetLastError'
⎕cse 'Close'
```

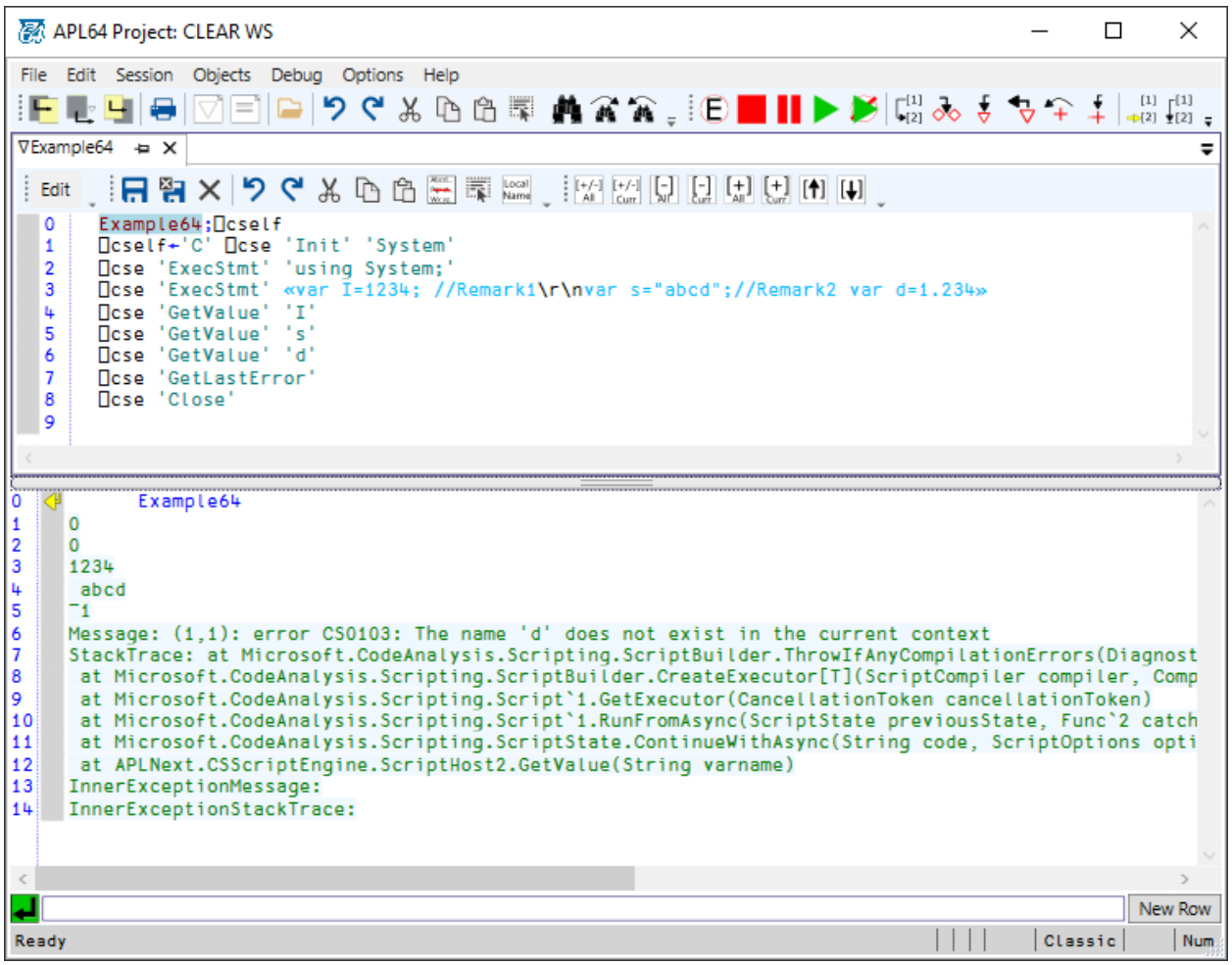


Example #64:

If a C# remark is included in the right argument to the CSE 'ExecStmt' method, any C# executable statement following that remark will be considered part of the remark and not executed unless it is separated by a new line character. In this example the C# escaped newline characters '\r\n' are used as the separator.

```
Example64;⎕cself
⎕cself←'C' ⎕cse 'Init' 'System'
```

```
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' «var I=1234; //Remark1\r\nvar s="abcd";//Remark2 var d=1.234»
⎕cse 'GetValue' 'I'
⎕cse 'GetValue' 's'
⎕cse 'GetValue' 'd'
⎕cse 'GetLastError '
⎕cse 'Close'
```

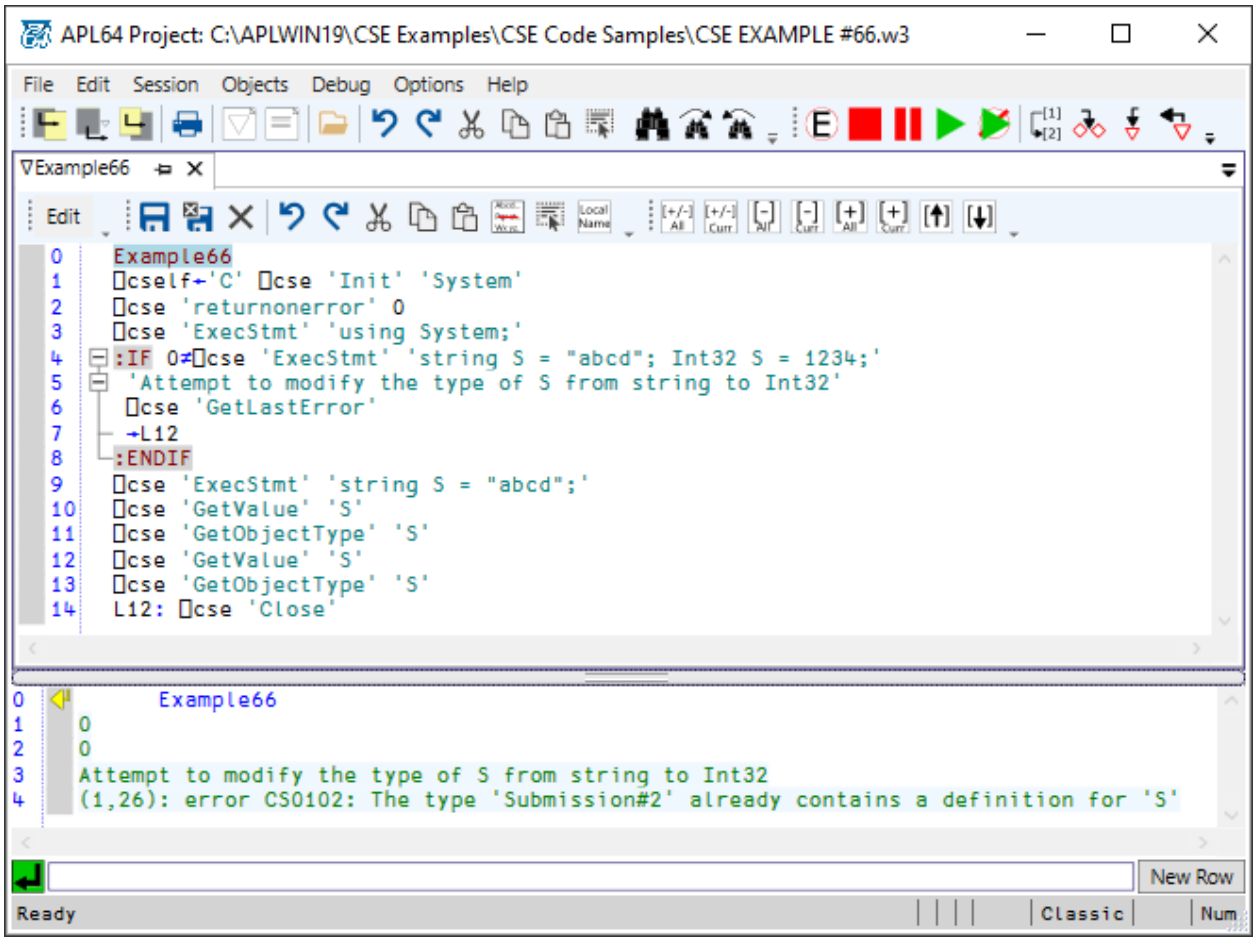


Example #66

In this example an attempt to modify the type of a C# variable in the same ExecStmt script fails.

```
Example66
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
⎕cse 'ExecStmt' 'using System;'
:IF 0≠⎕cse 'ExecStmt' 'string S = "abcd"; Int32 S = 1234;'
 'Attempt to modify the type of S from string to Int32'
```

```
 □cse 'GetLastError'
 →L12
:ENDIF
□cse 'ExecStmt' 'string S = "abcd";'
□cse 'GetValue' 'S'
□cse 'GetObjectType' 'S'
□cse 'GetValue' 'S'
□cse 'GetObjectType' 'S'
L12: □cse 'Close'
```



Value substitution of APL64 values

C# supports composite formatting of strings. This feature is enabled in the CSE 'ExecStmt' and 'GetValue' methods so that the C# executable statement which is the argument to these CSE methods can include 'format items'. Format items are indicated in the C# executable statement by enclosing braces and a format item index number, e.g. '{0}'. For each distinct format item number in a C# executable statement, there must be a corresponding APL64 value argument following in order after the C# executable statement argument.

Value substitution of APL64 values is optional for the CSE 'ExecStmt' and 'GetValue' methods.

Value Substitution syntax for the CSE 'ExecStmt' method:
... □cse 'ExecStmt' 'C# Executable Statement with format items;' (additional APL64 arguments)

Value Substitution syntax for the CSE 'GetValue' method:
... □cse 'GetValue' 'C# Executable Statement with format items' (additional APL64 arguments)

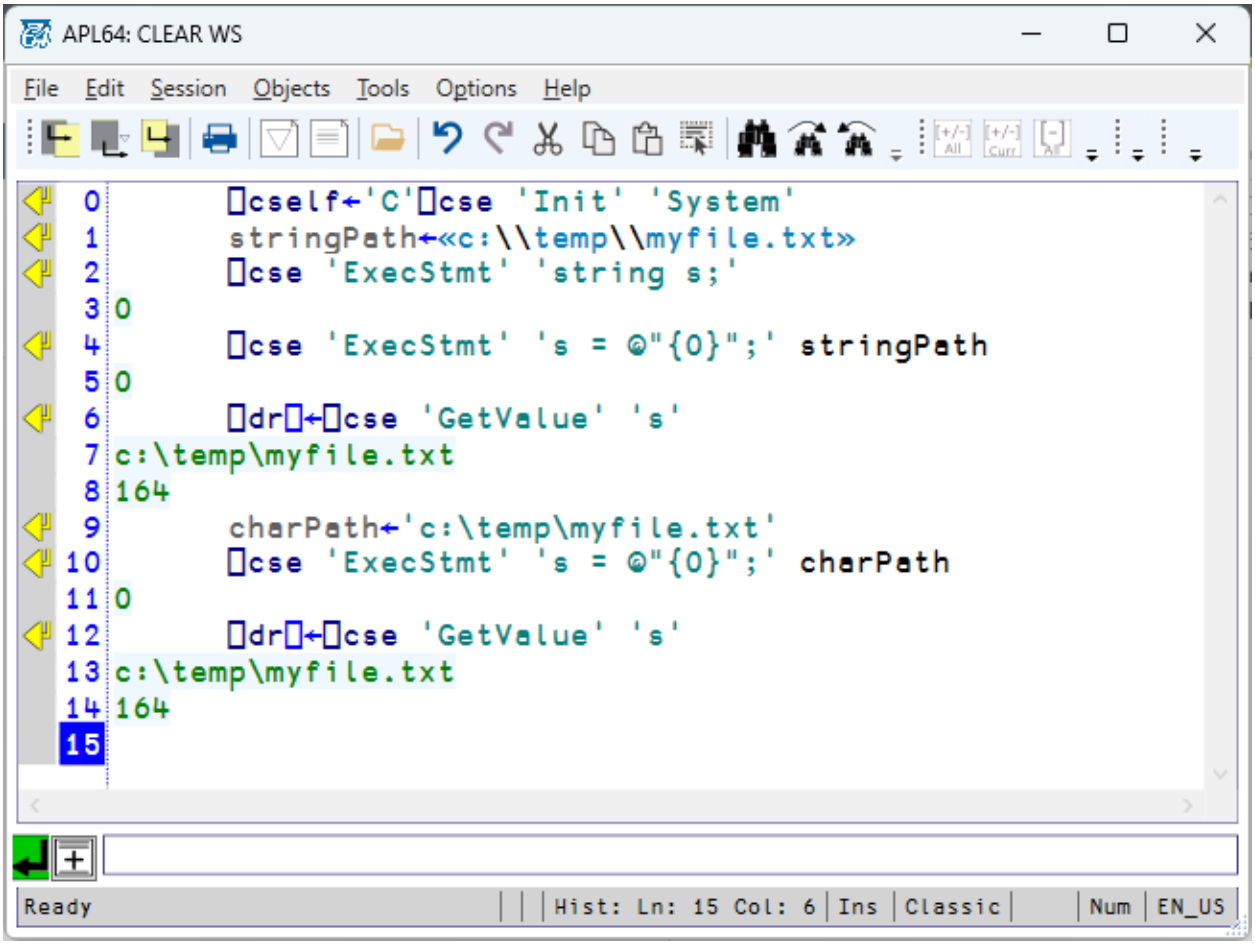
This value substitution feature is a means to prepare a C# executable statement. After the C# executable statement is prepared by substituting the APL64 values, that C# executable statement is executed by the .Net Common Language Runtime (CLR) without direct interaction with APL64. After CLR execution is complete, APL64 can query the results using CSE methods.

When the APL64 value to be substituted is a number, there is no potential ambiguity when the APL64 numeric value is substituted into the C# executable statement.

When the APL64 value to be substituted is a character scalar or vector there is the potential for the APL64 programmer to create an ambiguity because the APL64 text value might refer to a C# object name or be simply C# character or string data. To avoid this ambiguity:

- An APL64 substitution value which refers to C# object name is enclosed in quotes, e.g. "myCSVariable" or 'myCSVariable'.
- An APL64 substitution value which is simply a single C# character value, e.g. the C# character 'x' is provided by APL64 as "'x'".
- An APL64 substitution value which is simply a C# string value, e.g. the C# string "xyz" string is provided by APL64 as '"xys"'.
- Recall that characters and strings are not identical data types in .Net. Single quotes are used to enclose a .Net character and quotation marks are used to enclose the elements of a .Net string.
- Provide the appropriate quotes or quotation marks in the C# executable statement instead of the APL64-provided substitution value.

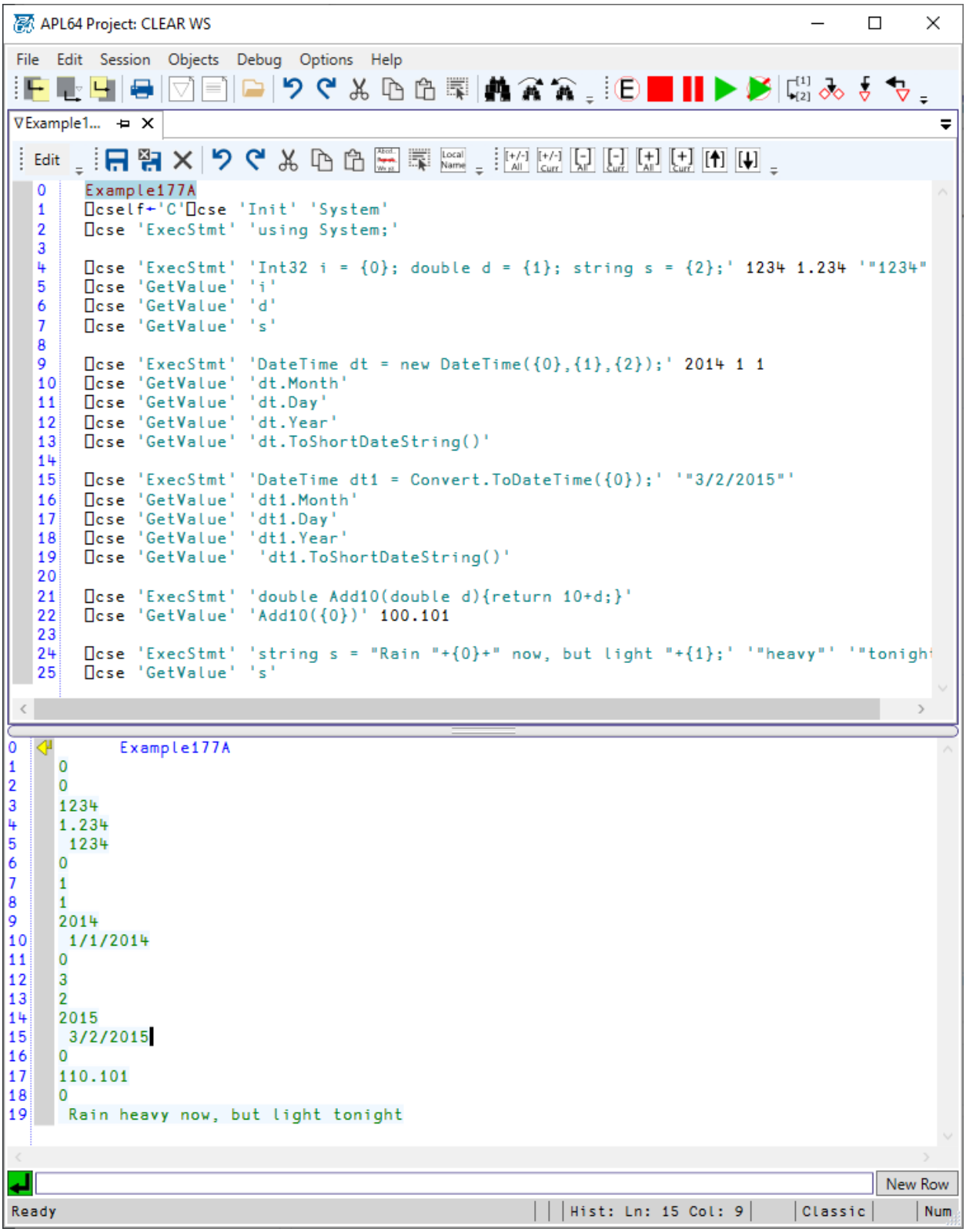
Example #177
A .Net string file path can be set using CSE value substitution:



Example #177A

```
Example177A
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'

⎕cse 'ExecStmt' 'Int32 i = {0}; double d = {1}; string s = {2};' 1234 1.234 '"1234"'
⎕cse 'GetValue' 'i'
⎕cse 'GetValue' 'd'
⎕cse 'GetValue' 's'

⎕cse 'ExecStmt' 'DateTime dt = new DateTime({0},{1},{2});' 2014 1 1
⎕cse 'GetValue' 'dt.Month'
⎕cse 'GetValue' 'dt.Day'
⎕cse 'GetValue' 'dt.Year'
⎕cse 'GetValue' 'dt.ToShortDateString()'
```

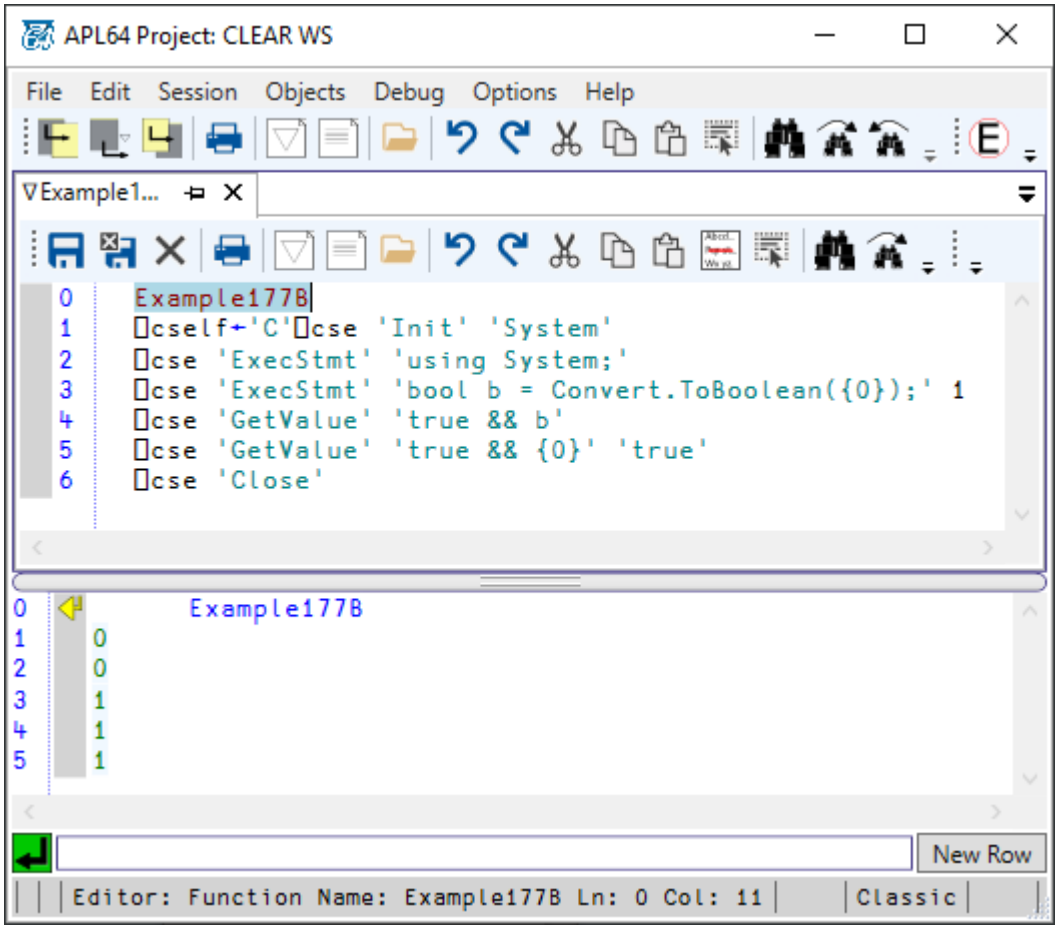
```
]cse 'ExecStmt' 'DateTime dt1 = Convert.ToDateTime({0});' '"3/2/2015"'
]cse 'GetValue' 'dt1.Month'
]cse 'GetValue' 'dt1.Day'
]cse 'GetValue' 'dt1.Year'
]cse 'GetValue'  'dt1.ToShortDateString()'

]cse 'ExecStmt' 'double Add10(double d){return 10+d;}'
]cse 'GetValue' 'Add10({0})' 100.101

]cse 'ExecStmt' 'string s = "Rain "+{0}+" now, but light "+{1};' '"heavy"' '"tonight"'
]cse 'GetValue' 's'
```



Example #177B

Using value substitution and the CSE instance ExecStmt method to set the value of a .Net Boolean.

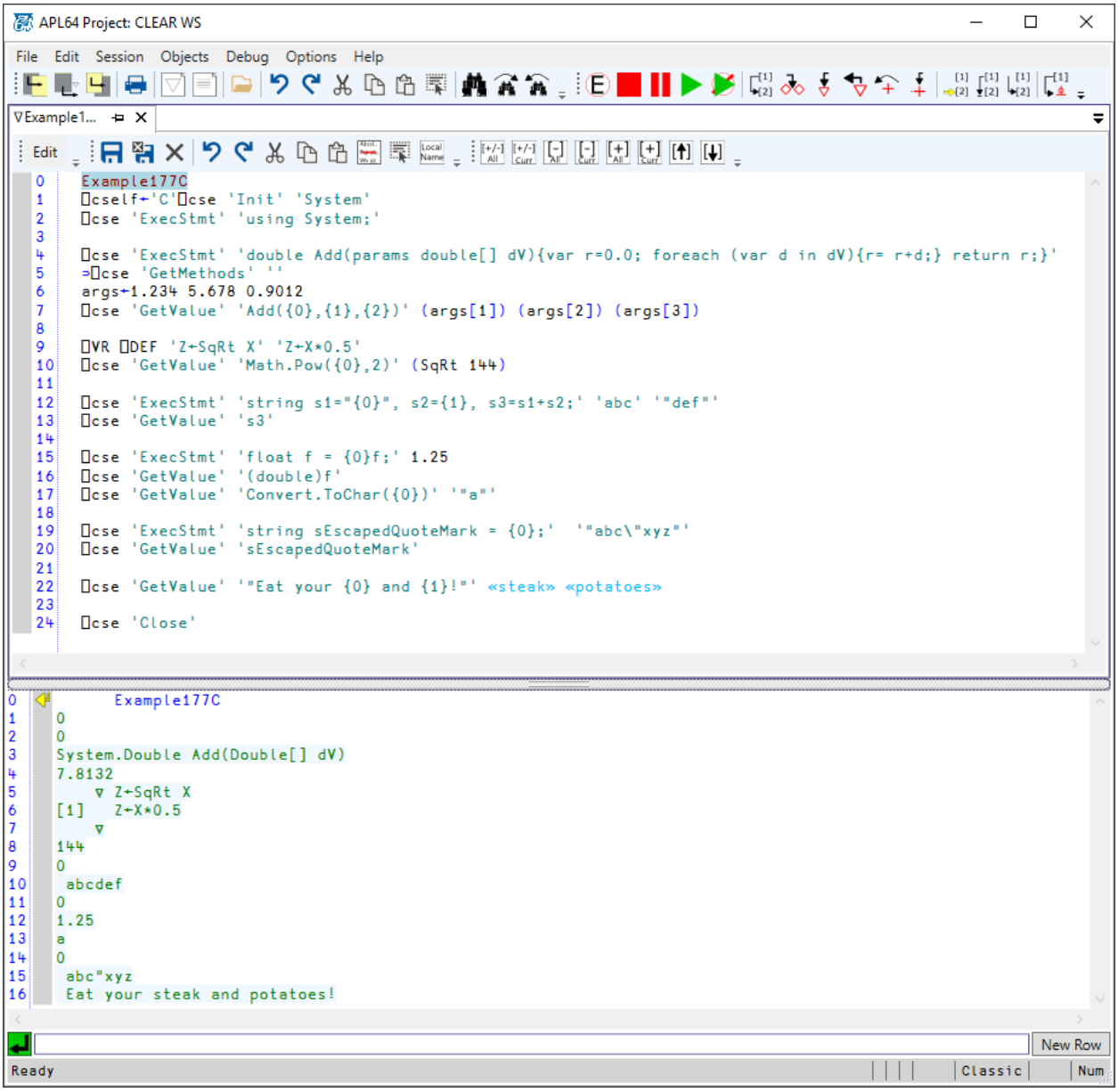
```
Example177B
 cself←'C' cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'
 cse 'ExecStmt' 'bool b = Convert.ToBoolean({0});' 1
 cse 'GetValue' 'true && b'
 cse 'GetValue' 'true && {0}' 'true'
 cse 'Close'
```



Example #177C

```
Example177C
 cself←'C' cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'

 cse 'ExecStmt' 'double Add(params double[] dV){var r=0.0; foreach (var d in dV){r= r+d;} return r;}'
⊃ cse 'GetMethods' ''
args←1.234 5.678 0.9012
 cse 'GetValue' 'Add({0},{1},{2})' (args[1]) (args[2]) (args[3])
```

```
⎕VR ⎕DEF 'Z←SqRt X' 'Z←X*0.5'
⎕cse 'GetValue' 'Math.Pow({0},2)' (SqRt 144)

⎕cse 'ExecStmt' 'string s1="{0}", s2={1}, s3=s1+s2;' 'abc' '"def"'
⎕cse 'GetValue' 's3'

⎕cse 'ExecStmt' 'float f = {0}f;' 1.25
⎕cse 'GetValue' '(double)f'
⎕cse 'GetValue' 'Convert.ToChar({0})' '"a"'

⎕cse 'ExecStmt' 'string sEscapedQuoteMark = {0};'  '"abc\"xyz"'
⎕cse 'GetValue' 'sEscapedQuoteMark'

⎕cse 'GetValue' '"Eat your {0} and {1}!"' «steak» «potatoes»

⎕cse 'Close'
```

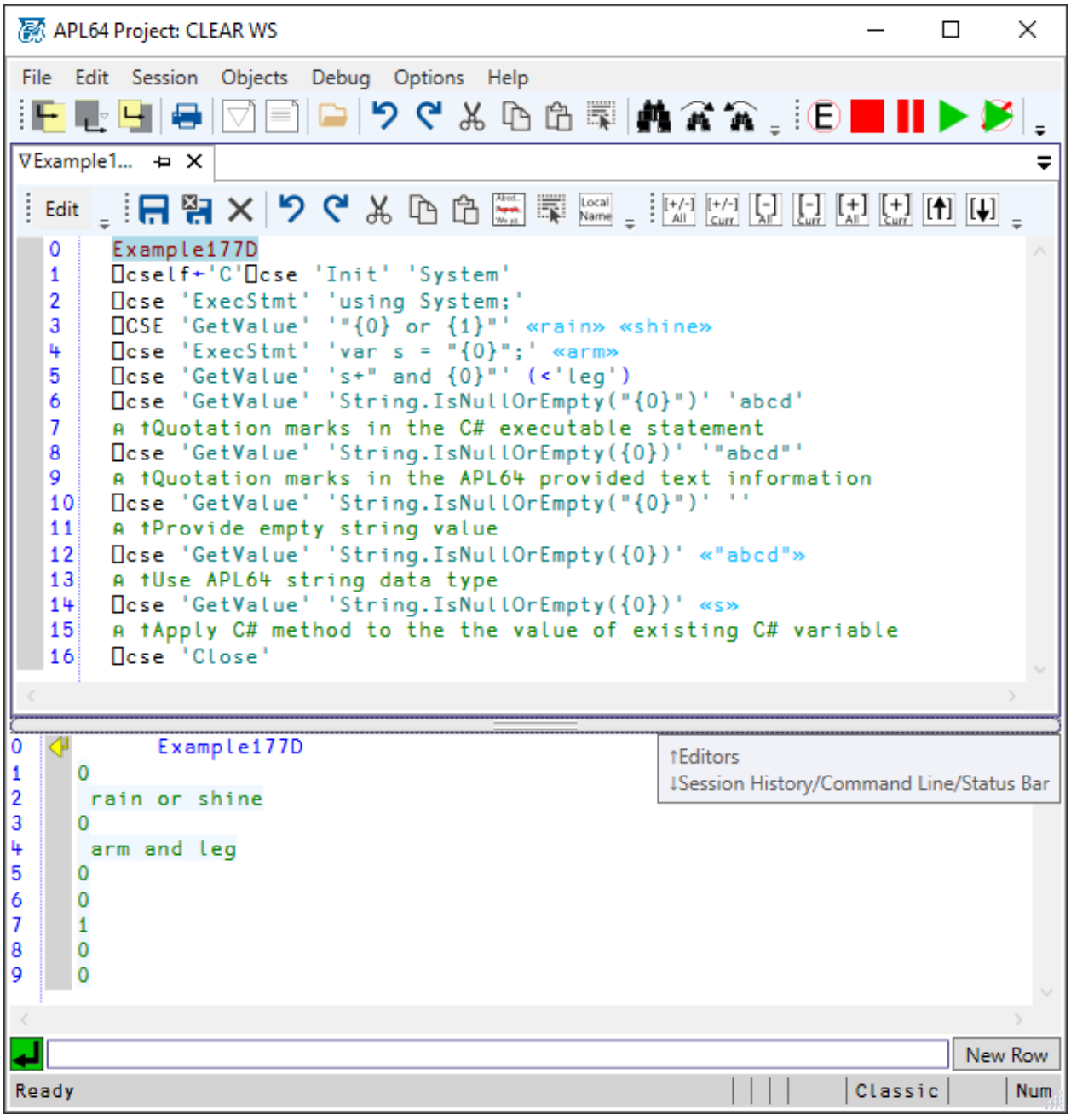


Example #177D

This example illustrates alternate ways to use value substitution of character data.

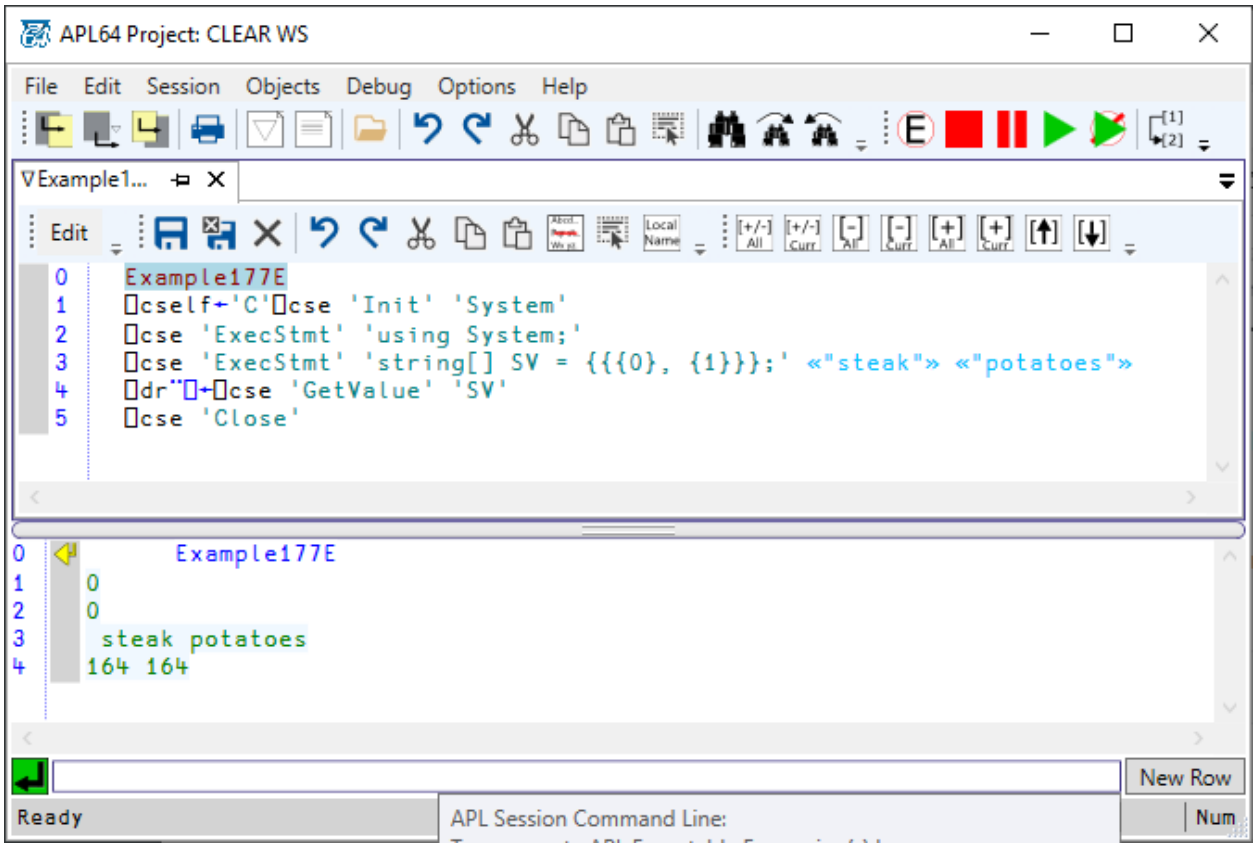
```
Example177D
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕CSE 'GetValue' '"{0} or {1}"' «rain» «shine»
⎕cse 'ExecStmt' 'var s = "{0}";' «arm»
⎕cse 'GetValue' 's+" and {0}"' (<'leg')
⎕cse 'GetValue' 'String.IsNullOrEmpty("{0}")' 'abcd'
⍝ ↑Quotation marks in the C# executable statement
```

```
]cse 'GetValue' 'String.IsNullOrEmpty({0})' '"abcd"'
⍝ ↑Quotation marks in the APL64 provided text information
]cse 'GetValue' 'String.IsNullOrEmpty("{0}")' ''
⍝ ↑Provide empty string value
]cse 'GetValue' 'String.IsNullOrEmpty({0})' «"abcd"»
⍝ ↑Use APL64 string data type
]cse 'GetValue' 'String.IsNullOrEmpty({0})' «s»
⍝ ↑Apply C# method to the the value of existing C# variable
]cse 'Close'
```



Example #177E

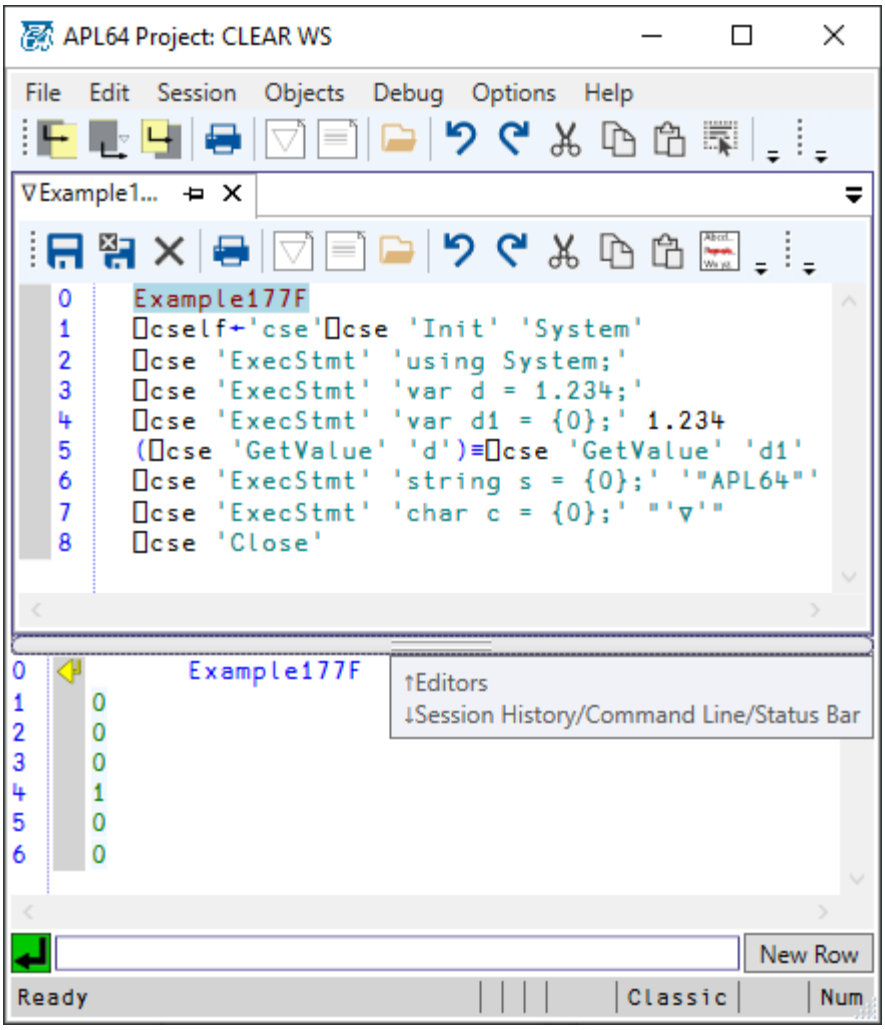
```
Example177E
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'string[] SV = {{{0}, {1}}};' «"steak"» «"potatoes"»
□dr¨□←□cse 'GetValue' 'SV'
□cse 'Close'
```



Example #177F

Value substitution with the CSE instance ExecStmt method.

```
Example177F
□cself←'cse'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'var d = 1.234;'
□cse 'ExecStmt' 'var d1 = {0};' 1.234
(□cse 'GetValue' 'd')≡□cse 'GetValue' 'd1'
□cse 'ExecStmt' 'string s = {0};' '"APL64"'
□cse 'ExecStmt' 'char c = {0};' "'∇'"
□cse 'Close'
```

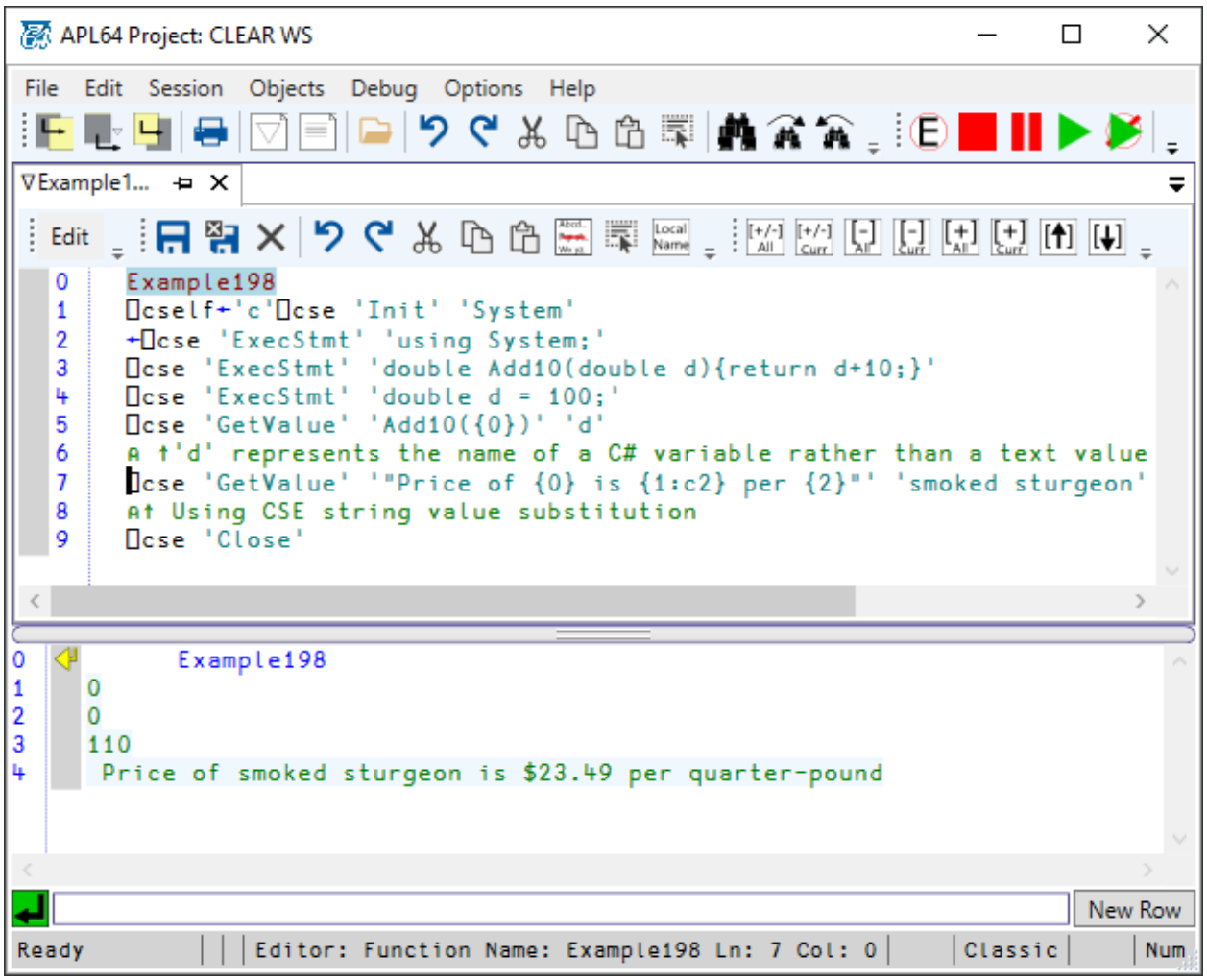


Example #198

In this example APL64 text scalars or vectors are the values being substituted. The example illustrates:

- Use value substitution to apply a C# method to an existing C# variable where the substituted text information represents the name of the C# variable
- Use C# formatting options in when substituting a value

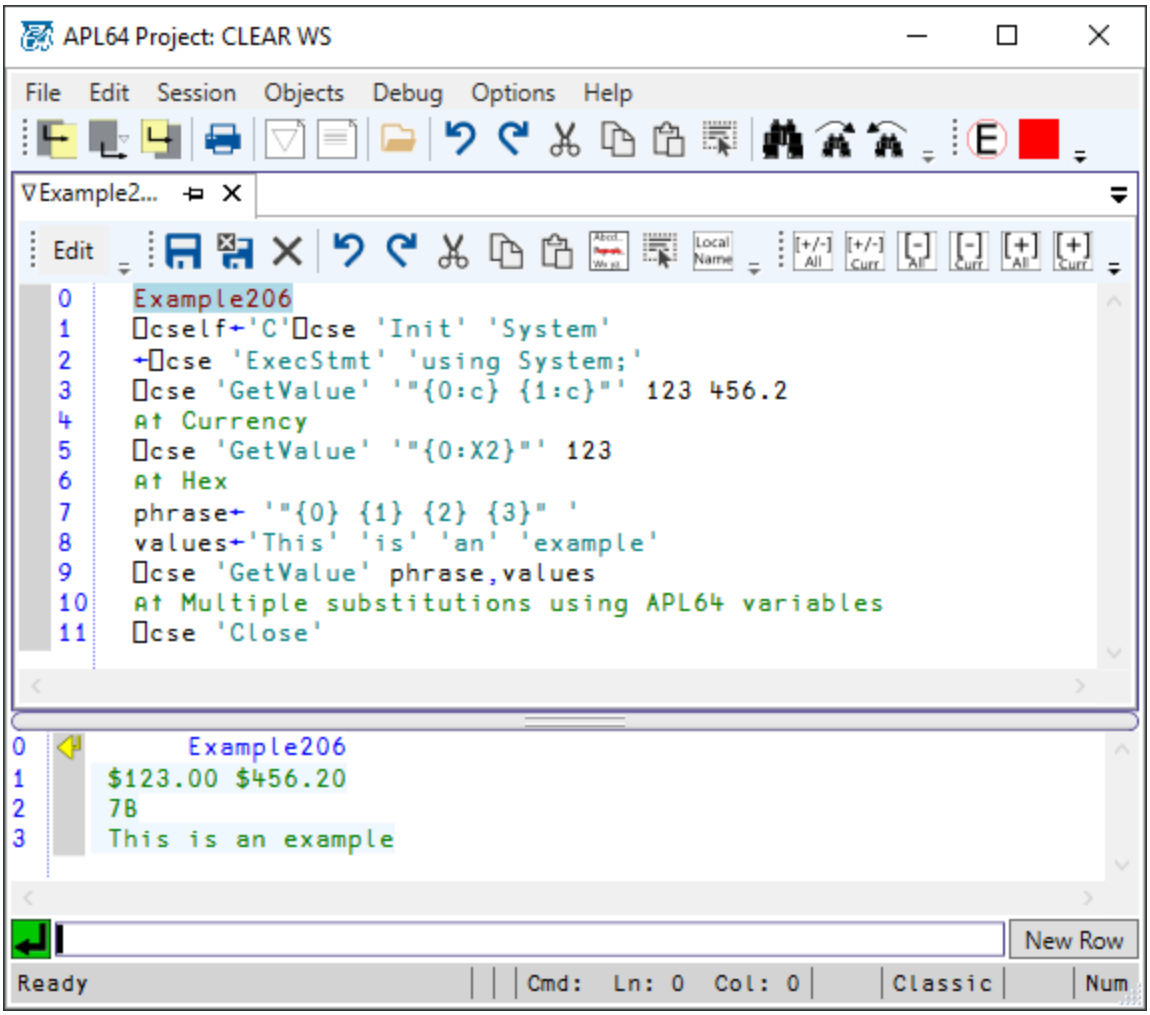
```
Example198
⎕cself←'c'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;' ⎕cse 'ExecStmt' 'double Add10(double d){return d+10;}'
⎕cse 'ExecStmt' 'double d = 100;'
⎕cse 'GetValue' 'Add10({0})' 'd'
⍝ ↑'d' represents the name of a C# variable rather than a text value
⎕cse 'GetValue' '"Price of {0} is {1:c2} per {2}"' 'smoked sturgeon' 23.49 'quarter-pound'
⍝↑ Using CSE string value substitution
⎕cse 'Close'
```



Example #206

This example illustrates value substitution using .Net Format Extensions.

```
Example206
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'GetValue' '"{0:c} {1:c}"' 123 456.2
⍝↑ Currency
⎕cse 'GetValue' '"{0:X2}"' 123
⍝↑ Hex
phrase← '"{0} {1} {2} {3}" '
values←'This' 'is' 'an' 'example'
⎕cse 'GetValue' phrase,values
⍝↑ Multiple substitutions using APL64 variables
⎕cse 'Close'
```



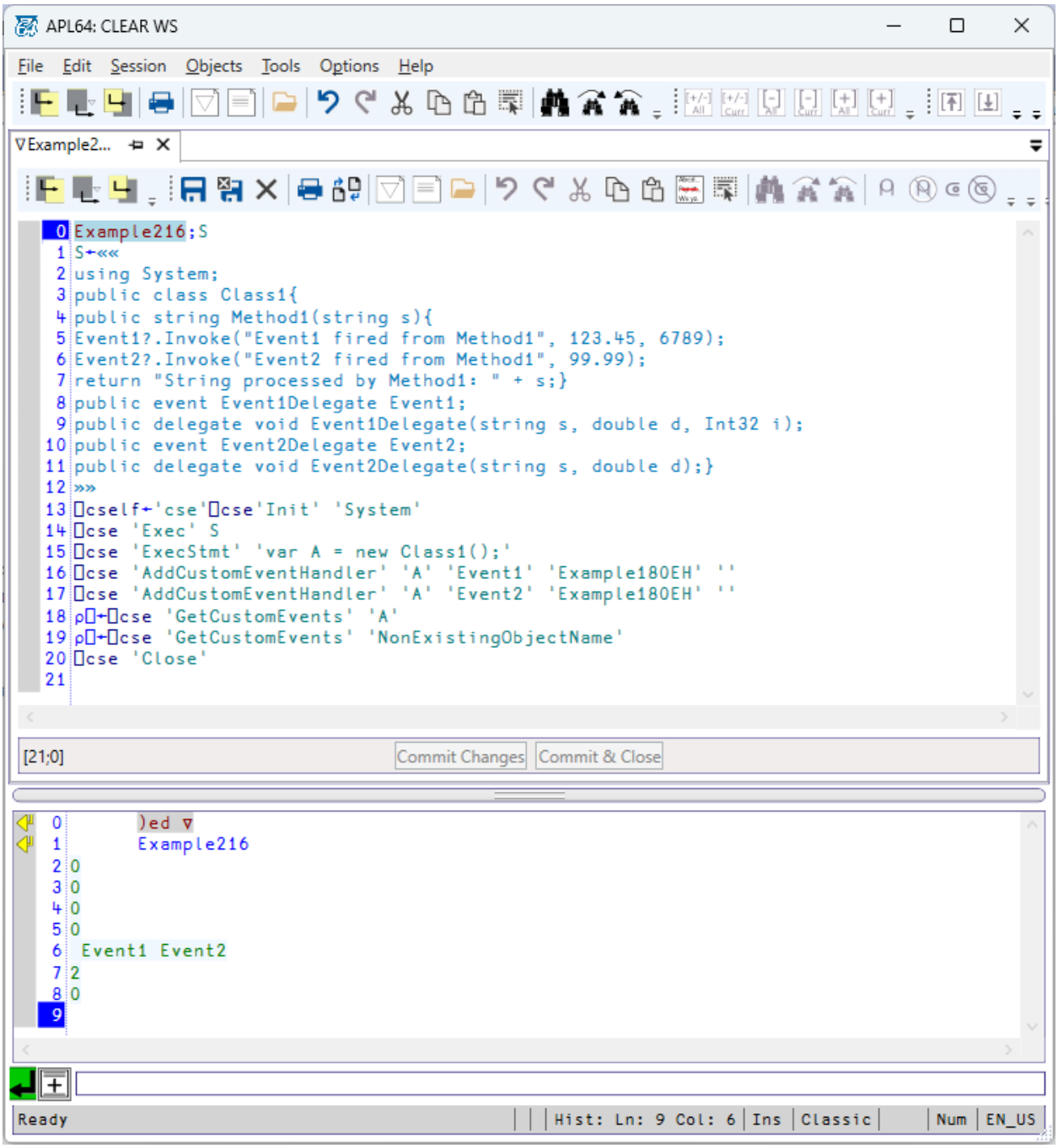
CSE GetCustomEvents Method

The CSE 'GetCustomEvents' method returns a vector of character vectors containing the names of the custom events exposed by the C# object specified in the right argument to the CSE 'GetCustomEvents' method which have been subscribed using the CSE 'AddCustomEventHandler' or 'AddCustomEventHandlerEx' methods.

Example #216

```
Example216;S
S←««
using System;
public class Class1{
public string Method1(string s){
Event1?.Invoke("Event1 fired from Method1", 123.45, 6789);
Event2?.Invoke("Event2 fired from Method1", 99.99);
return "String processed by Method1: " + s;}
public event Event1Delegate Event1;
```

```
public delegate void Event1Delegate(string s, double d, Int32 i);
public event Event2Delegate Event2;
public delegate void Event2Delegate(string s, double d);}
»»
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
⎕cse 'ExecStmt' 'var A = new Class1();'
⎕cse 'AddCustomEventHandler' 'A' 'Event1' 'Example180EH' ''
⎕cse 'AddCustomEventHandler' 'A' 'Event2' 'Example180EH' ''
⍴⎕←⎕cse 'GetCustomEvents' 'A'
⍴⎕←⎕cse 'GetCustomEvents' 'NonExistingObjectName'
⎕cse 'Close'
```



CSE GetEvents Method

This method returns a vector of character vectors containing the names of the public .Net events which are defined on the specified .Net object.

Required Right Argument of the CSE 'GetEvents' method:

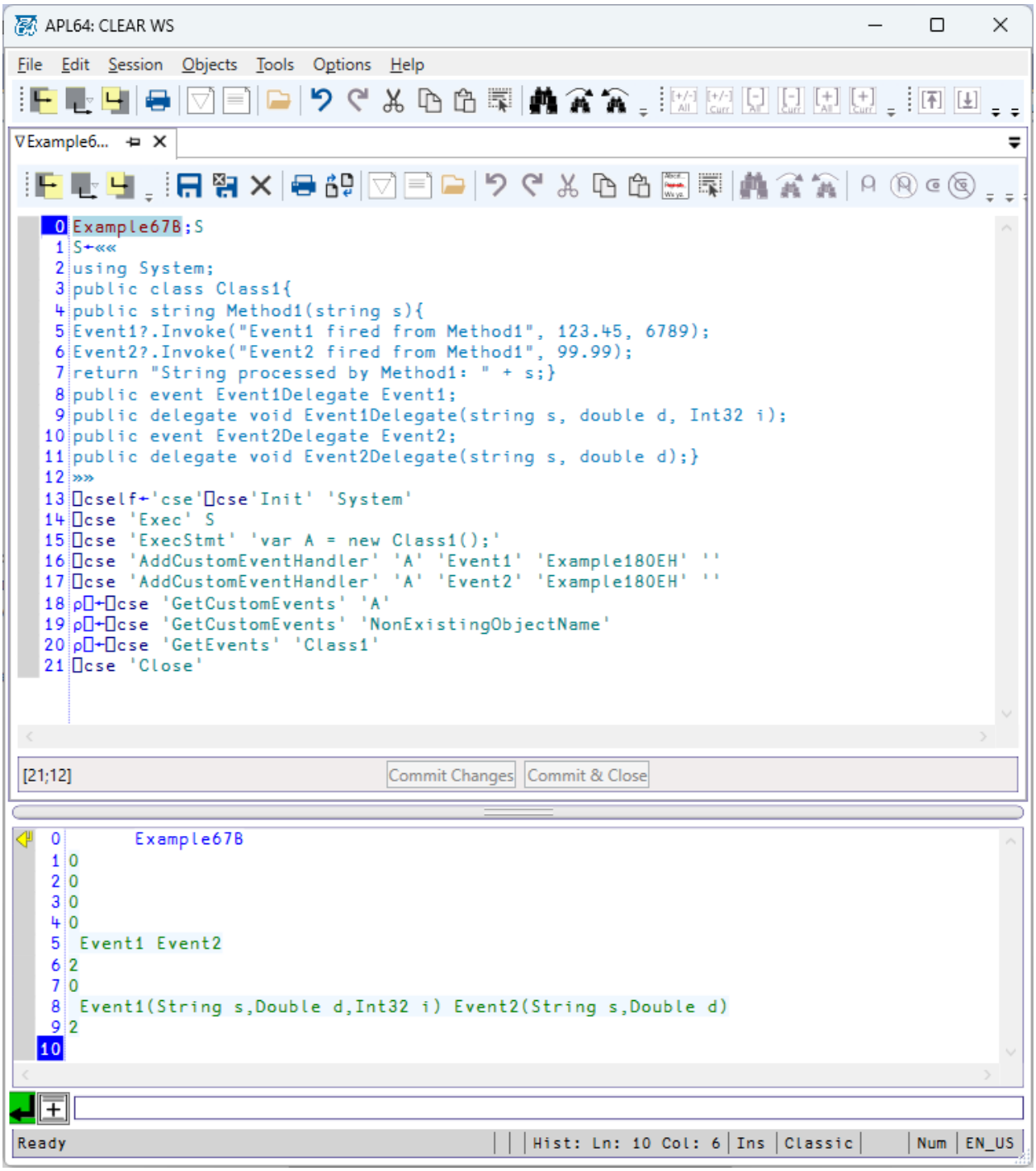
- Public events can be defined within the scope of a .Net class.
    - If the class is an instance class, the required right argument of the CSE 'GetEvents' method is the C# name of an instance of that class.

- If the class is a static class, the required right argument of the CSE 'GetEvents' method is the class name.

- Although it is also possible to define a public event within the scope of a CSE instance, this is not recommended because APL64 is a single-threaded application, so the firing of such an event cannot be handled in APL64. Therefore, the CSE 'GetEvents' method is not designed to report public events which have been defined within the scope of a CSE instance.

Example #67B
In this example custom events are exposed by Class1 and are reported by the GetEvents method.

```
Example67B;S
S←««
using System;
public class Class1{
public string Method1(string s){
Event1?.Invoke("Event1 fired from Method1", 123.45, 6789);
Event2?.Invoke("Event2 fired from Method1", 99.99);
return "String processed by Method1: " + s;}
public event Event1Delegate Event1;
public delegate void Event1Delegate(string s, double d, Int32 i);
public event Event2Delegate Event2;
public delegate void Event2Delegate(string s, double d);}
»»
□cself←'cse'□cse'Init' 'System'
□cse 'Exec' S
□cse 'ExecStmt' 'var A = new Class1();'
□cse 'AddCustomEventHandler' 'A' 'Event1' 'Example180EH' ''
□cse 'AddCustomEventHandler' 'A' 'Event2' 'Example180EH' ''
ρ□←□cse 'GetCustomEvents' 'A'
ρ□←□cse 'GetCustomEvents' 'NonExistingObjectName'
ρ□←□cse 'GetEvents' 'Class1'
□cse 'Close'
```

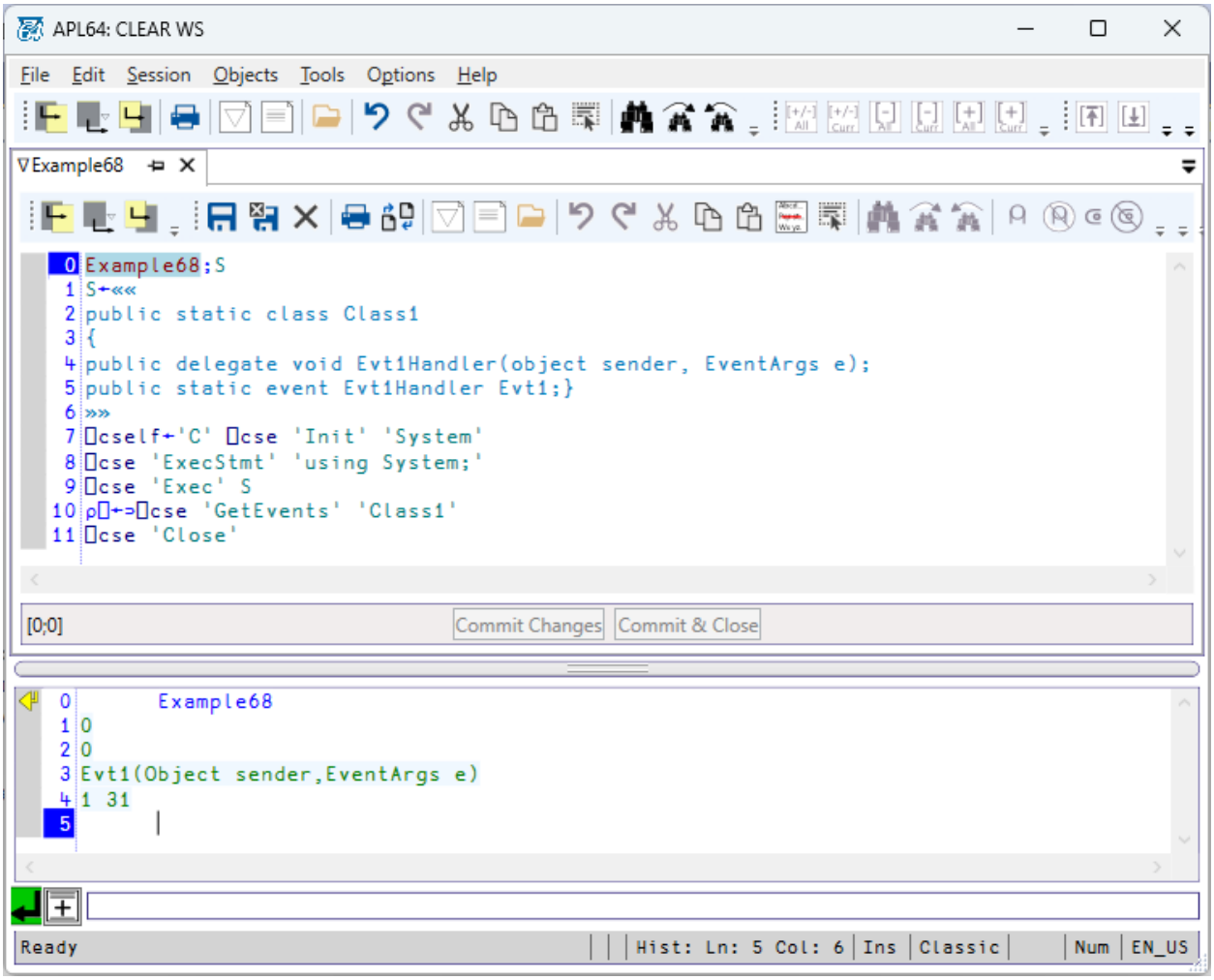


Example #68:

To obtain the public events of a static class, use the class name as the right argument to the CSE 'GetEvents' method.

```
Example68
S←'public static class Class1{'
```

```
S←⎕OVERV S 'public delegate void Evt1Handler(object sender, EventArgs e);'
S←⎕OVERV S 'public static event Evt1Handler Evt1;}'
S

⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'Exec' S
⍴⎕←⊃⎕cse 'GetEvents' 'Class1'
⎕cse 'Close'
```



CSE GetLastError Method

The CSE 'GetLastError' method is used to retrieve the most recent error message, if any, generated by the C# debugger or compiler for a specified instance of the CSE object and return that text as a character vector to APL64. Multiple lines of the error message are separated by a new line character.

The CSE ‘GetLastError’ method has an option argument indicating the number of lines of the error message to return with syntax: ⎕cse 'GetLastError' [#lines]

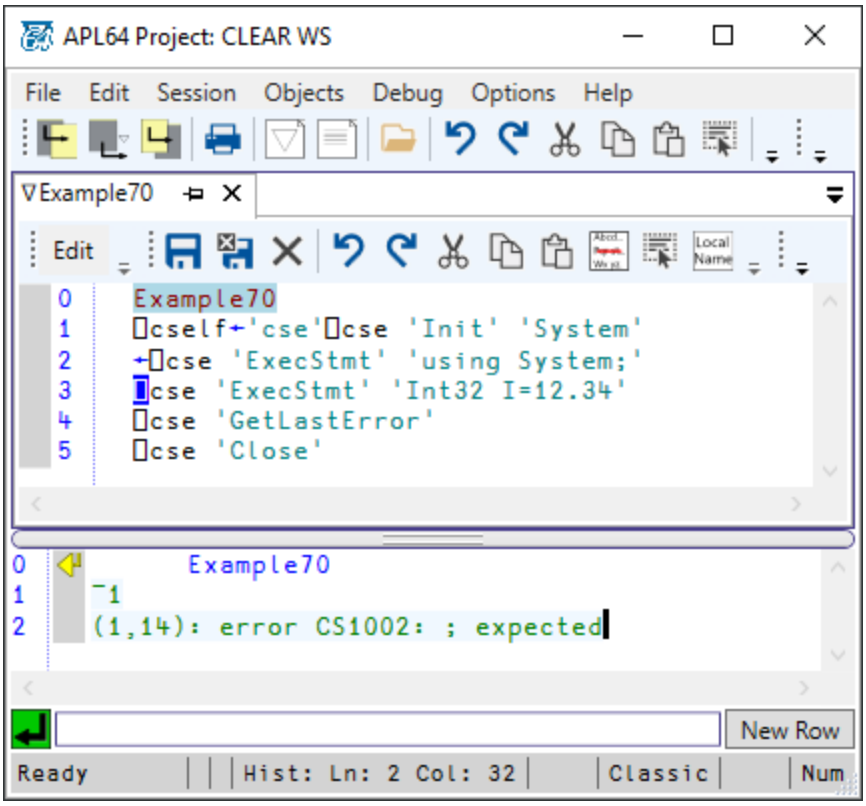
When the CSE reports an error message provided by the C# debugger or complier, the message including standard information:

- (row,column) in index origin one (1) of the location in the CSE script which triggered the C# exception.
- The CS#### error number which can be searched in the Microsoft Developer Network (MSDN) documentation on-line.
- The text of the C# error message which can also be searched in the MSDN documentation.
- The text of a C# error message is not translated to APL-style error message text, so that searches in MSDN are easily performed. Error messages with the format (row, column): error CS####: … arise because of errors in the programmer-provide CSE script when it is analyzed by the C# debugger/compiler.
- Such errors are reported one at a time.

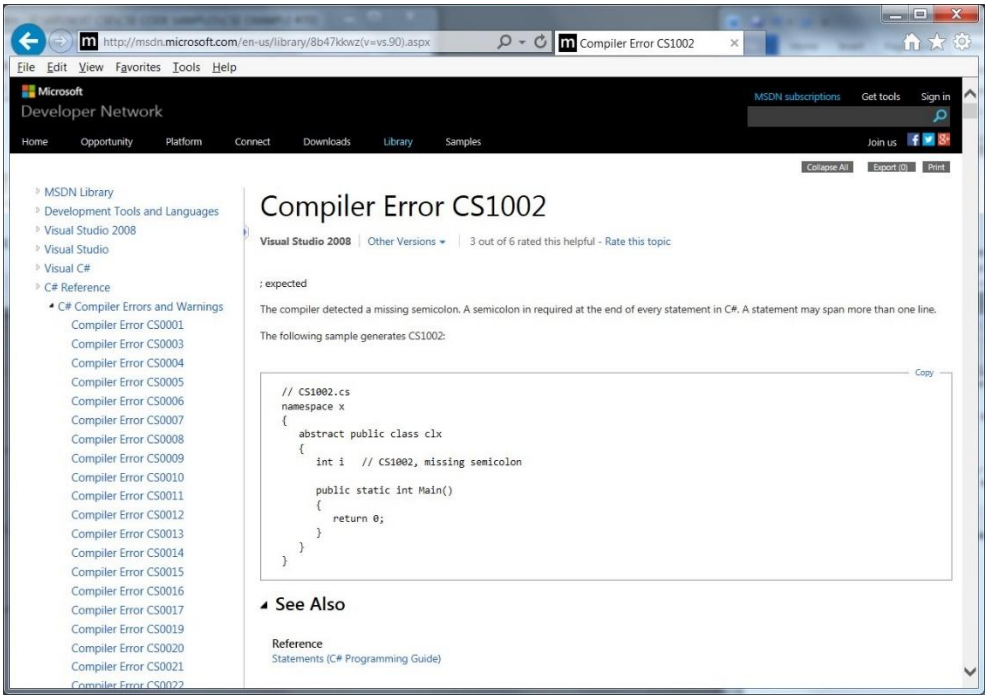
When the CSE client or CSE engine reports an error the (row,column) information is not present. This is because such an error is not caused by a C# script, so the C# debugger and compiler are not involved. The C# exception error message includes the exception message, exception stack trace, inner exception message, if any, and inner exception stack trace, if any,

Example #70:
In this example the C# debugger identifies an invalid C# statement in the CSE script by row and column number. The Microsoft C# debugger identified the missing C# semi-colon statement separator in row #1 and column #13 [both values using C# index origin 1] of the programmer-provided CSE script executed via the CSE ‘ExecStmt’ method. The missing ‘;’ statement separator exception in the programmer-provided script is reported first, even though the attempt to assign a double value to an integer-typed variable occurs earlier in the script. This example illustrates that for CSE scripts, exceptions are reported one at a time and exceptions are not necessarily reported in the order of the rows columns of the CSE script.

```
Example70
⎕cself←'cse'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'Int32 I=12.34'
⎕cse 'GetLastError'
⎕cse 'Close'
```



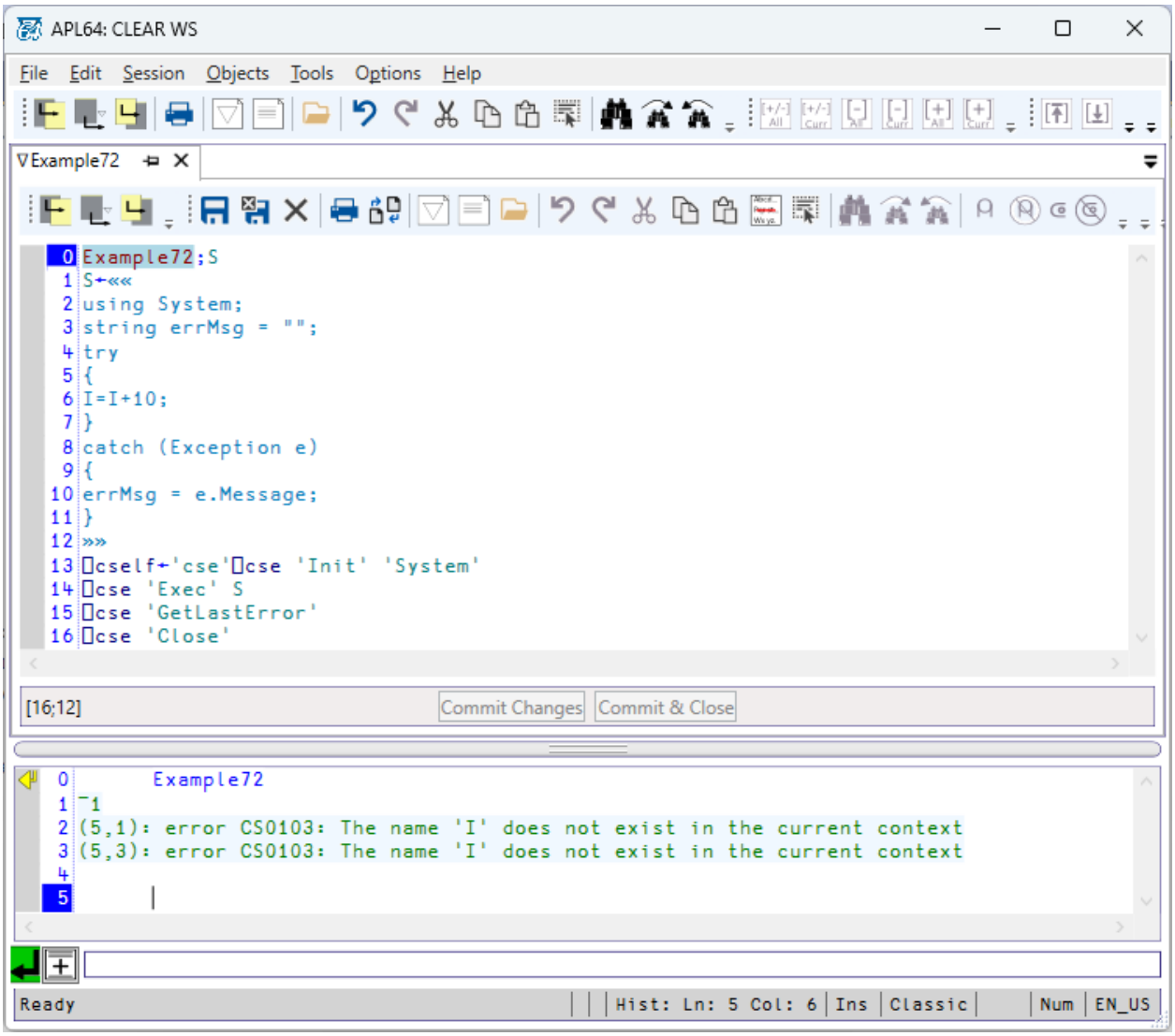
The results of a search in MSDN yields:



Example #72:

In this example the APL64 programmer has incorporated a C# 'try/catch' structure to catch and report exceptions in a C# script, but the C# debugger identifies the error before execution of the script can occur.

```
Example72;S
S←««
using System;
string errMsg = "";
try
{
I=I+10;
}
catch (Exception e)
{
errMsg = e.Message;
}
»»
⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'Exec' S
⎕cse 'GetLastError'
⎕cse 'Close'
```

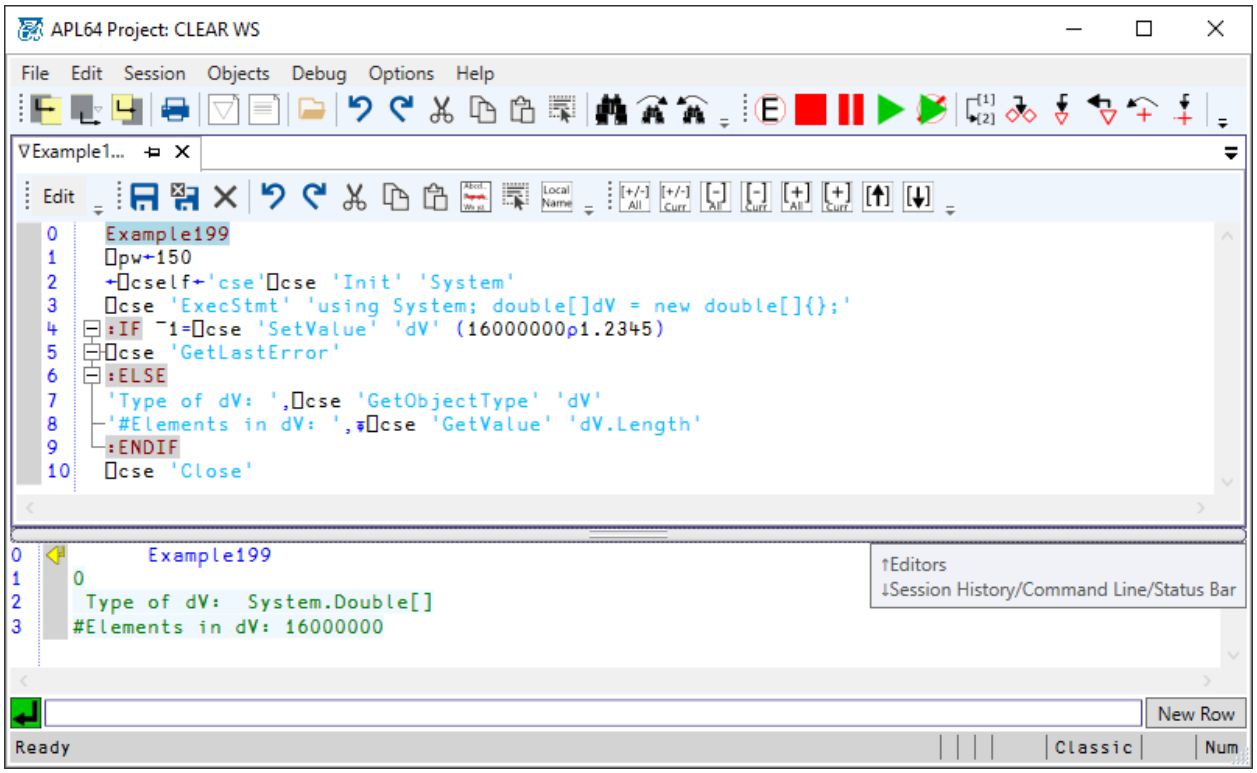


Example #199:

In this example a large data object is sent to the CSE instance, illustrating that processing time is affected by the data size. Depending on the available memory of the target workstation, an exception may be thrown indicating that the transfer of the data from APL64 to the CSE is not possible.

```
Example199
⎕pw←150
←⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System; double[]dV = new double[]{};'
:IF ¯1=⎕cse 'SetValue' 'dV' (16000000⍴1.2345)
⎕cse 'GetLastError'
:ELSE
'Type of dV: ',⎕cse 'GetObjectType' 'dV'
'#Elements in dV: ',⍕⎕cse 'GetValue' 'dV.Length'
:ENDIF
```

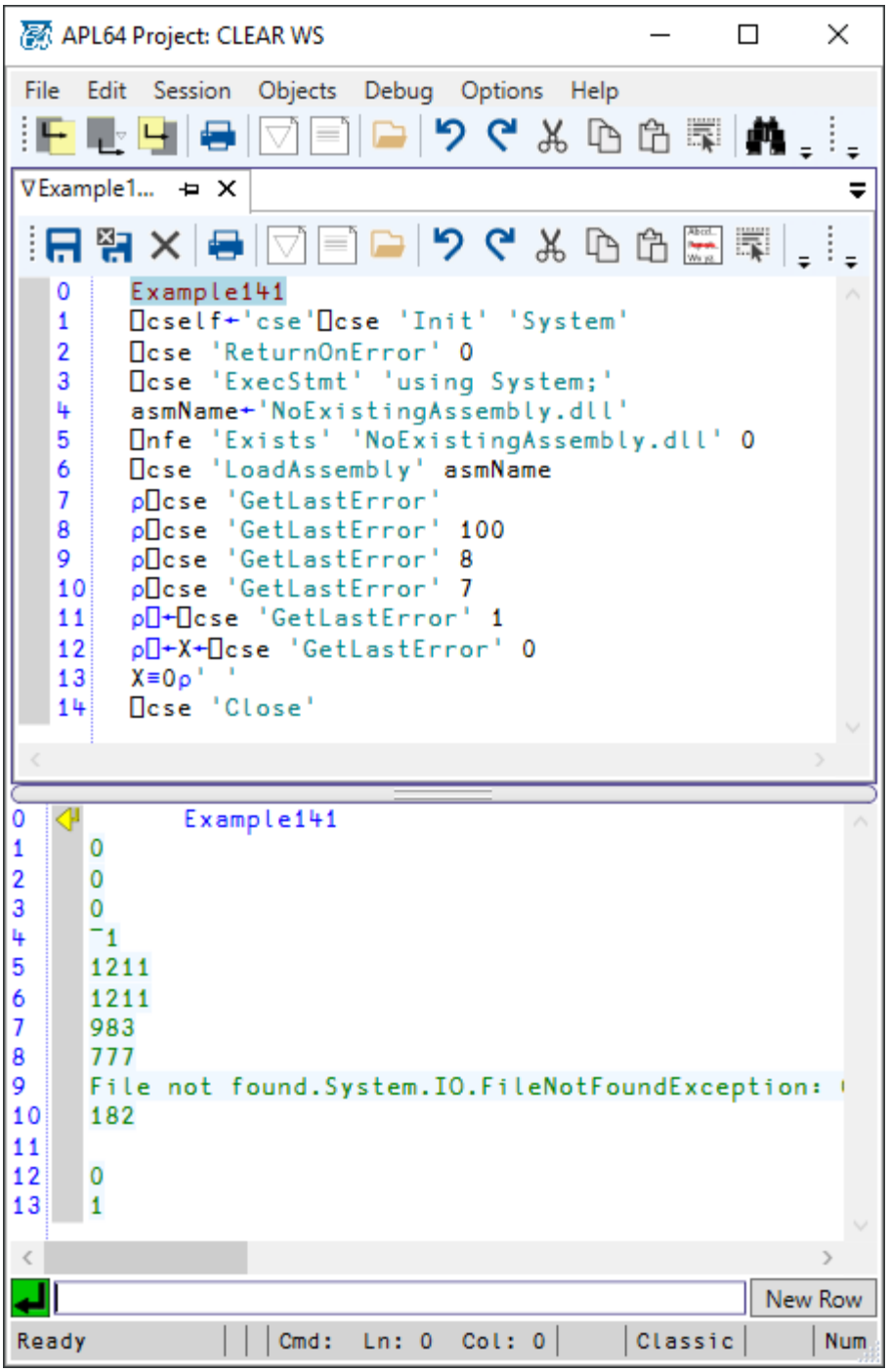
```
⎕cse 'Close'
```



Example #141

In this example the number of lines of the last error message to be returned by the CSE ‘GetLastError’ is specified by the APL64 programmer.

```
Example141
⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System;'
asmName←'NoExistingAssembly.dll'
⎕nfe 'Exists' 'NoExistingAssembly.dll' 0
⎕cse 'LoadAssembly' asmName
⍴⎕cse 'GetLastError'
⍴⎕cse 'GetLastError' 100
⍴⎕cse 'GetLastError' 8
⍴⎕cse 'GetLastError' 7
⍴⎕←⎕cse 'GetLastError' 1
⍴⎕←X←⎕cse 'GetLastError' 0
X≡0⍴' '
⎕cse 'Close'
```



CSE GetMethods Method

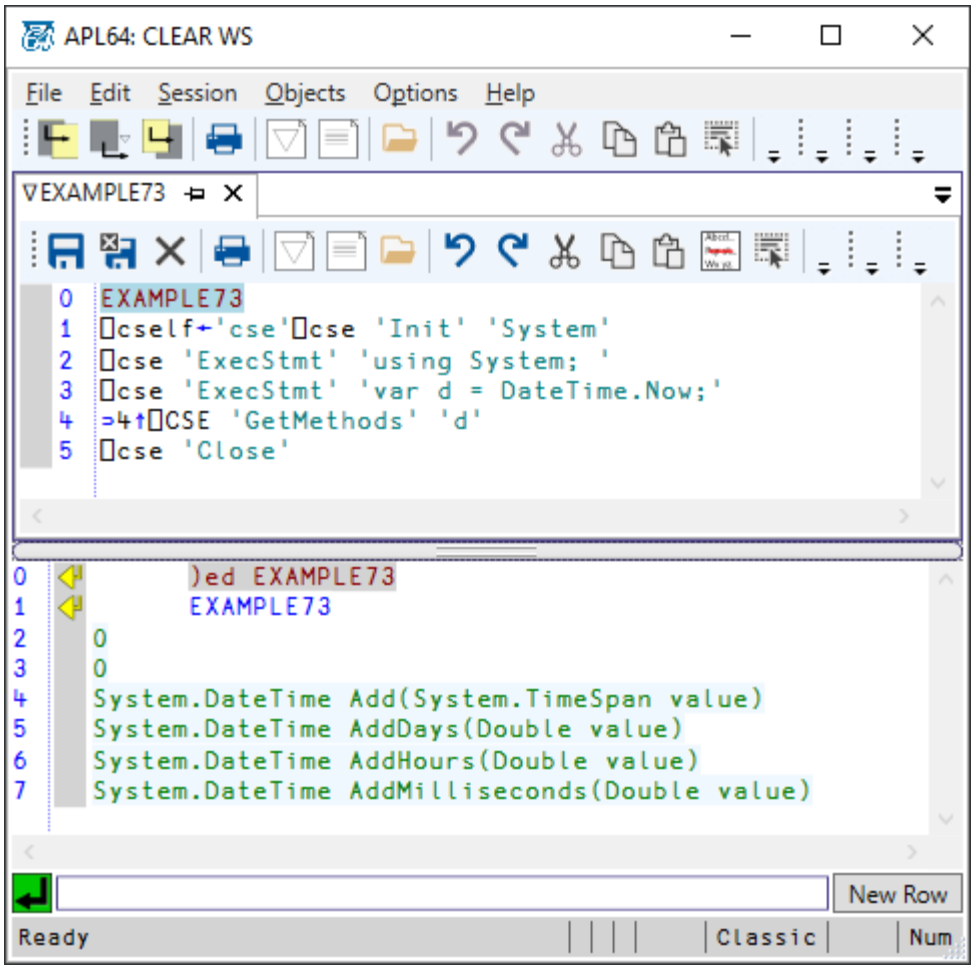
The CSE 'GetMethods' method returns a vector of text vectors containing the syntax of the public methods of the .Net object specified in the right argument of the CSE 'GetMethods' method. The results are sorted alphabetically in increasing order according to the .Net object's method name. Each element of the result of the CSE 'GetMethods' method includes a prefix indicating the type of .Net object (or void) returned by the .Net method.

Required Right Argument of the CSE 'GetMethods' method:

- For public methods defined within the scope of a .Net class.
    - The right argument of the CSE 'GetMethods' method should be the name of an instance of the class if the class is an instance class.
    - The right argument of the CSE 'GetMethods' method should be the name of the class if the class is a static class or if the instance class has static methods.
- For methods defined within the scope of a CSE instance, the required right argument of the CSE 'GetMethods' method is '' (i.e. 0⍴' ').
- The CSE 'GetMethods' method has two additional optional right arguments:
    - The 2nd Right argument is a Boolean 'isStatic' which when set to '1' will cause the CSE 'GetMethods' method to return the static methods. The default value is false (0).
    - The 3rd Right argument is a Boolean 'isPrivate' which when set to '1' will cause the CSE 'GetMethods' method to include private methods in the result. The default value is false (0).

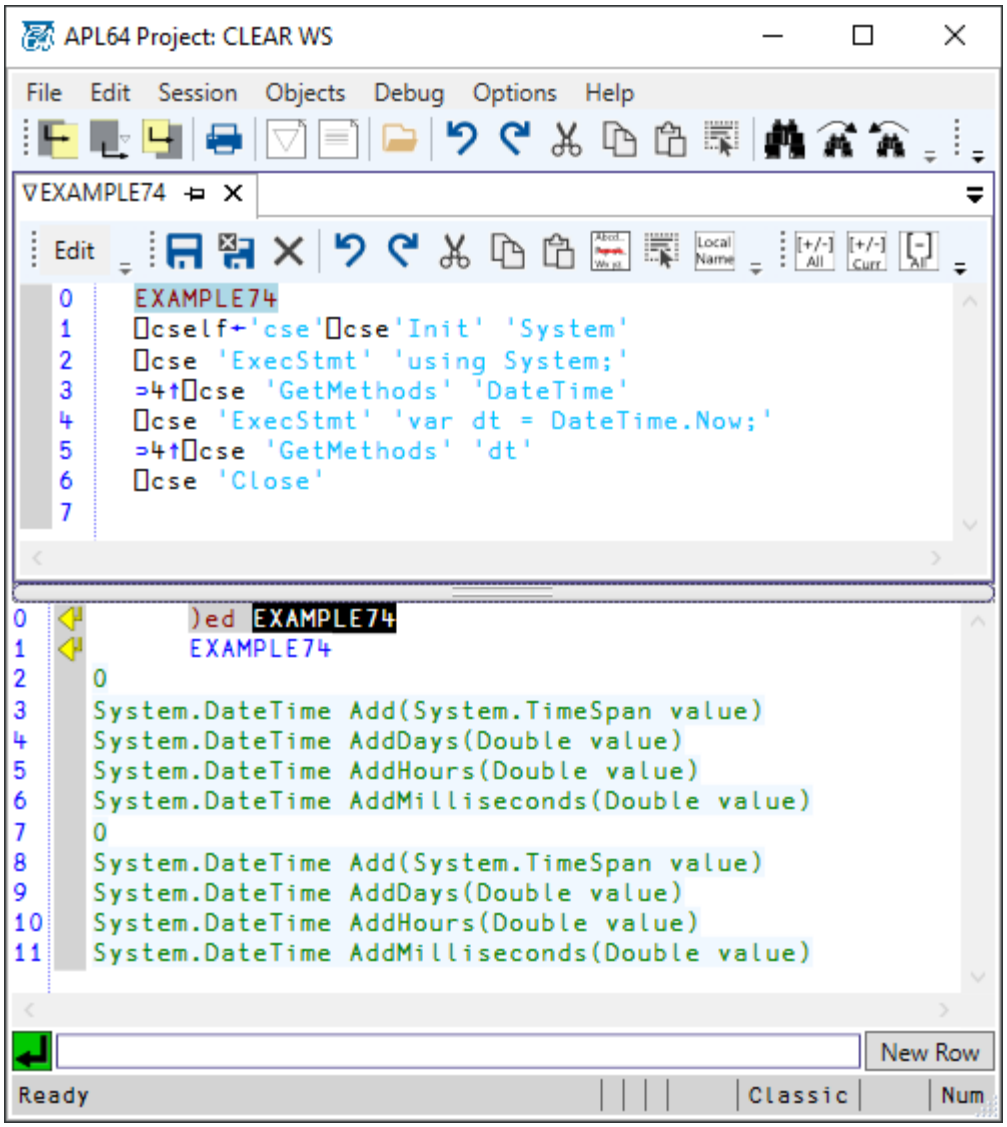
Example #73:
In this example, obtaining the methods of an instance of the System.DateTime class, is illustrated.

```
EXAMPLE73
⎕←cself←'cse'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System; '
⎕cse 'ExecStmt' 'var d = DateTime.Now;'
⊃4↑⎕CSE 'GetMethods' 'd'
⎕cse 'Close'
```



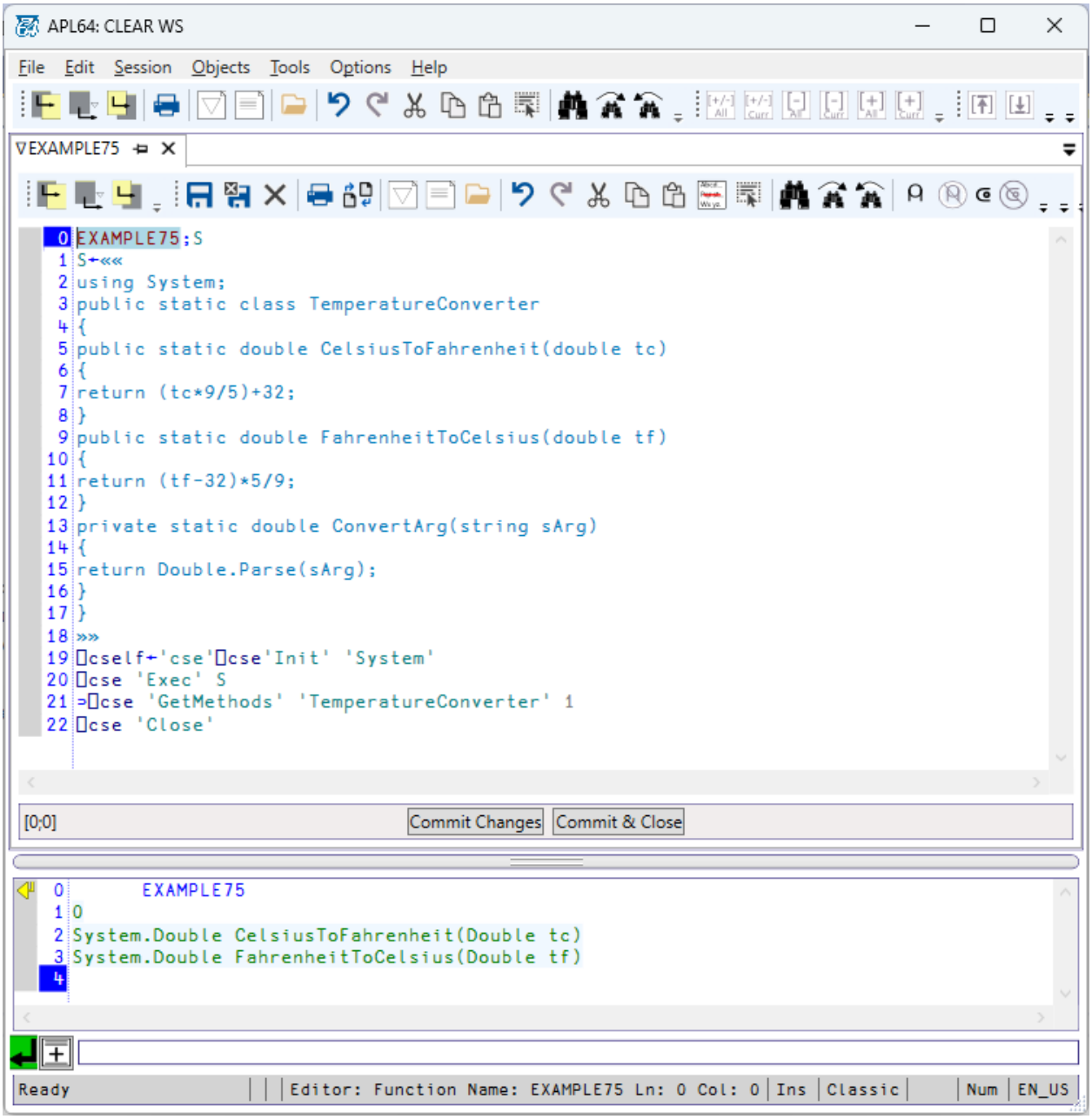
Example #74:
This example illustrates the use of the CSE 'GetMethods' method to obtain the methods of the 'DateTime' class and the method of an instance of the DateTime class.

```
EXAMPLE74
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⊃4↑⎕cse 'GetMethods' 'DateTime'
⎕cse 'ExecStmt' 'var dt = DateTime.Now;'
⊃4↑⎕cse 'GetMethods' 'dt'
⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
∇EXAMPLE74
Edit
0 EXAMPLE74
1 ⎕cself←'cse'⎕cse'Init' 'System'
2 ⎕cse 'ExecStmt' 'using System;'
3 ⍝4↑⎕cse 'GetMethods' 'DateTime'
4 ⎕cse 'ExecStmt' 'var dt = DateTime.Now;'
5 ⍝4↑⎕cse 'GetMethods' 'dt'
6 ⎕cse 'Close'
7
0)ed EXAMPLE74
1 EXAMPLE74
2 0
3 System.DateTime Add(System.TimeSpan value)
4 System.DateTime AddDays(Double value)
5 System.DateTime AddHours(Double value)
6 System.DateTime AddMilliseconds(Double value)
7 0
8 System.DateTime Add(System.TimeSpan value)
9 System.DateTime AddDays(Double value)
10 System.DateTime AddHours(Double value)
11 System.DateTime AddMilliseconds(Double value)
New Row
Ready
Classic
Num

Example #75:
In this example a custom static class, TemperatureConverter, is created using the CSE script ‘S’. Only the public methods of the TemperatureConverter class are returned by the CSE ‘GetMethods’ method.

```
EXAMPLE75;S
S←««
using System;
public static class TemperatureConverter
{
public static double CelsiusToFahrenheit(double tc)
{
return (tc*9/5)+32;
}
public static double FahrenheitToCelsius(double tf)
{
return (tf-32)*5/9;
}
private static double ConvertArg(string sArg)
{
return Double.Parse(sArg);
}
}
»»
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
⊃⎕cse 'GetMethods' 'TemperatureConverter' 1
⎕cse 'Close'
```

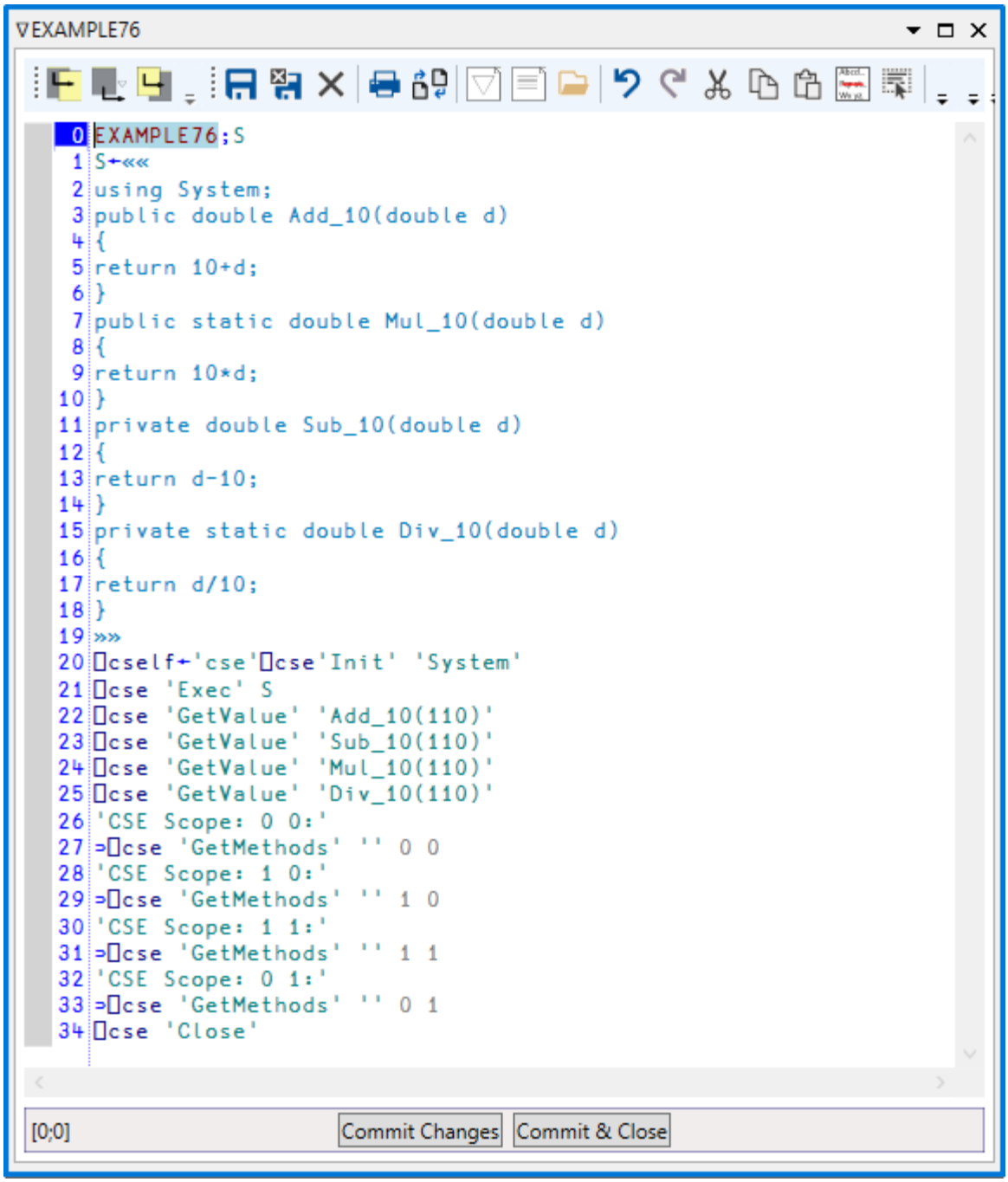


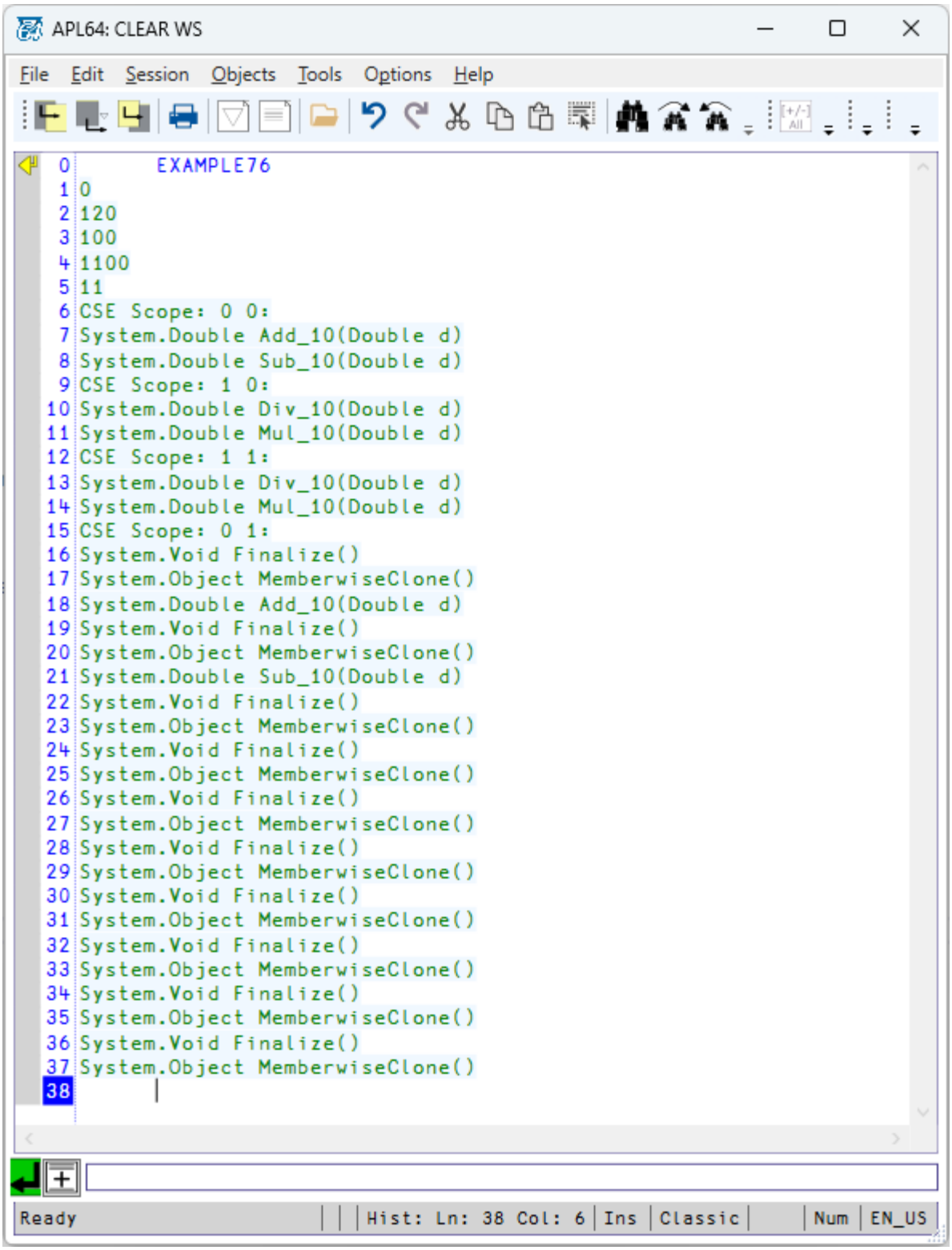
Example #76:

This example illustrates four methods defined in the scope of the CSE instance. Only public methods are reported by the CSE 'GetMethods' method, but all four methods are accessible in this scope despite the private attribute included in the definition of the 'Sub_10' and 'Div_10' methods. All variations of the 'GetMethods' arguments are illustrated. Some private methods of the CSE engine are also revealed in this example.

```
EXAMPLE76;S
S←««
using System;
```

```
public double Add_10(double d)
{
return 10+d;
}
public static double Mul_10(double d)
{
return 10*d;
}
private double Sub_10(double d)
{
return d-10;
}
private static double Div_10(double d)
{
return d/10;
}
»»
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
⎕cse 'GetValue' 'Add_10(110)'
⎕cse 'GetValue' 'Sub_10(110)'
⎕cse 'GetValue' 'Mul_10(110)'
⎕cse 'GetValue' 'Div_10(110)'
'CSE Scope: 0 0:'
⊃⎕cse 'GetMethods' '' 0 0
'CSE Scope: 1 0:'
⊃⎕cse 'GetMethods' '' 1 0
'CSE Scope: 1 1:'
⊃⎕cse 'GetMethods' '' 1 1
'CSE Scope: 0 1:'
⊃⎕cse 'GetMethods' '' 0 1
⎕cse 'Close'
```

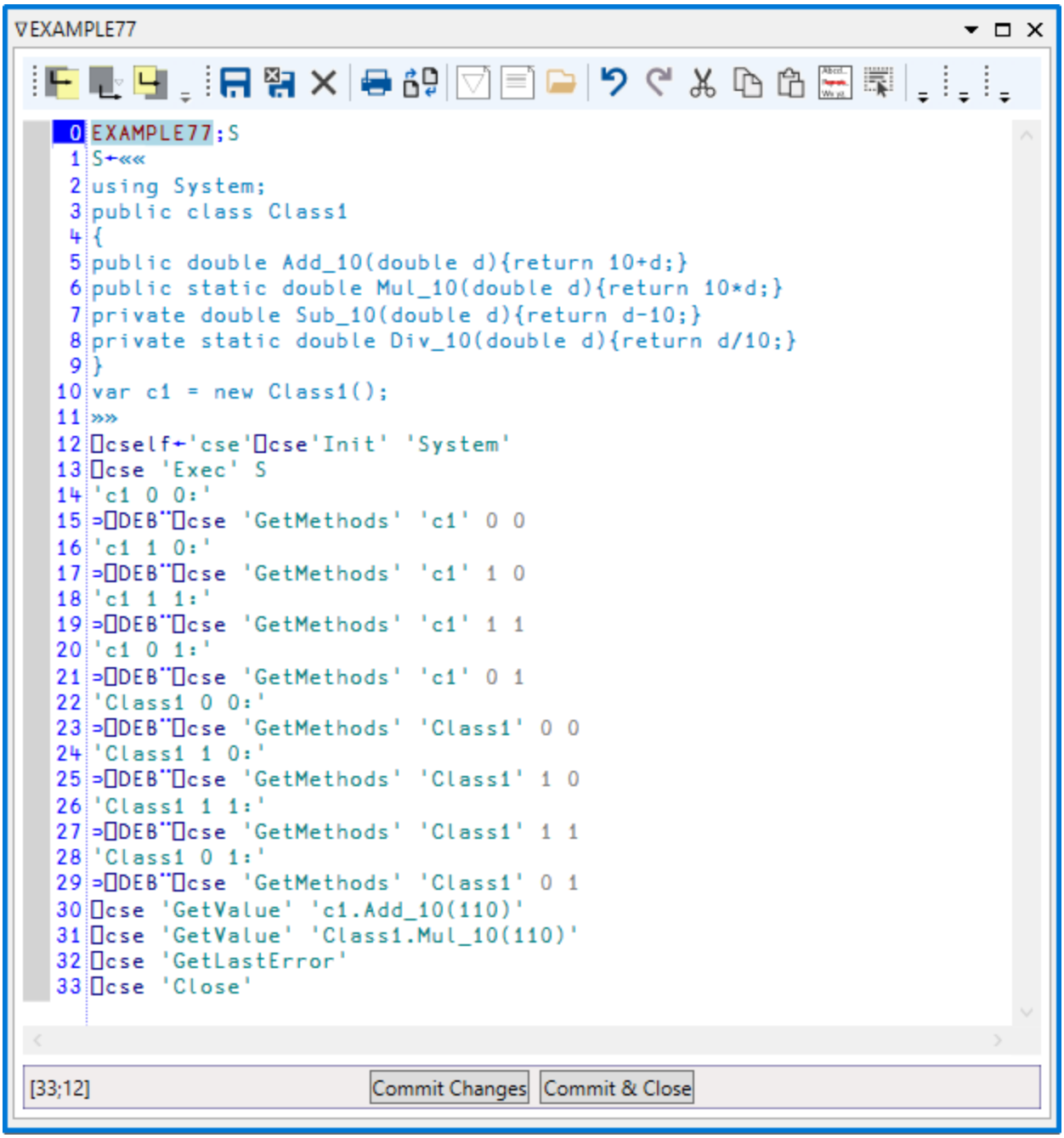

∇EXAMPLE76
0 EXAMPLE76;S
1 S←⍬⍬
2 using System;
3 public double Add_10(double d)
4 {
5 return 10+d;
6 }
7 public static double Mul_10(double d)
8 {
9 return 10*d;
10 }
11 private double Sub_10(double d)
12 {
13 return d-10;
14 }
15 private static double Div_10(double d)
16 {
17 return d/10;
18 }
19 ⍬⍬⍬
20 ⎕cself←'cse'⎕cse'Init' 'System'
21 ⎕cse 'Exec' S
22 ⎕cse 'GetValue' 'Add_10(110)'
23 ⎕cse 'GetValue' 'Sub_10(110)'
24 ⎕cse 'GetValue' 'Mul_10(110)'
25 ⎕cse 'GetValue' 'Div_10(110)'
26 'CSE Scope: 0 0:'
27 ⍝⎕cse 'GetMethods' '' 0 0
28 'CSE Scope: 1 0:'
29 ⍝⎕cse 'GetMethods' '' 1 0
30 'CSE Scope: 1 1:'
31 ⍝⎕cse 'GetMethods' '' 1 1
32 'CSE Scope: 0 1:'
33 ⍝⎕cse 'GetMethods' '' 0 1
34 ⎕cse 'Close'
[0;0]
Commit Changes
Commit & Close

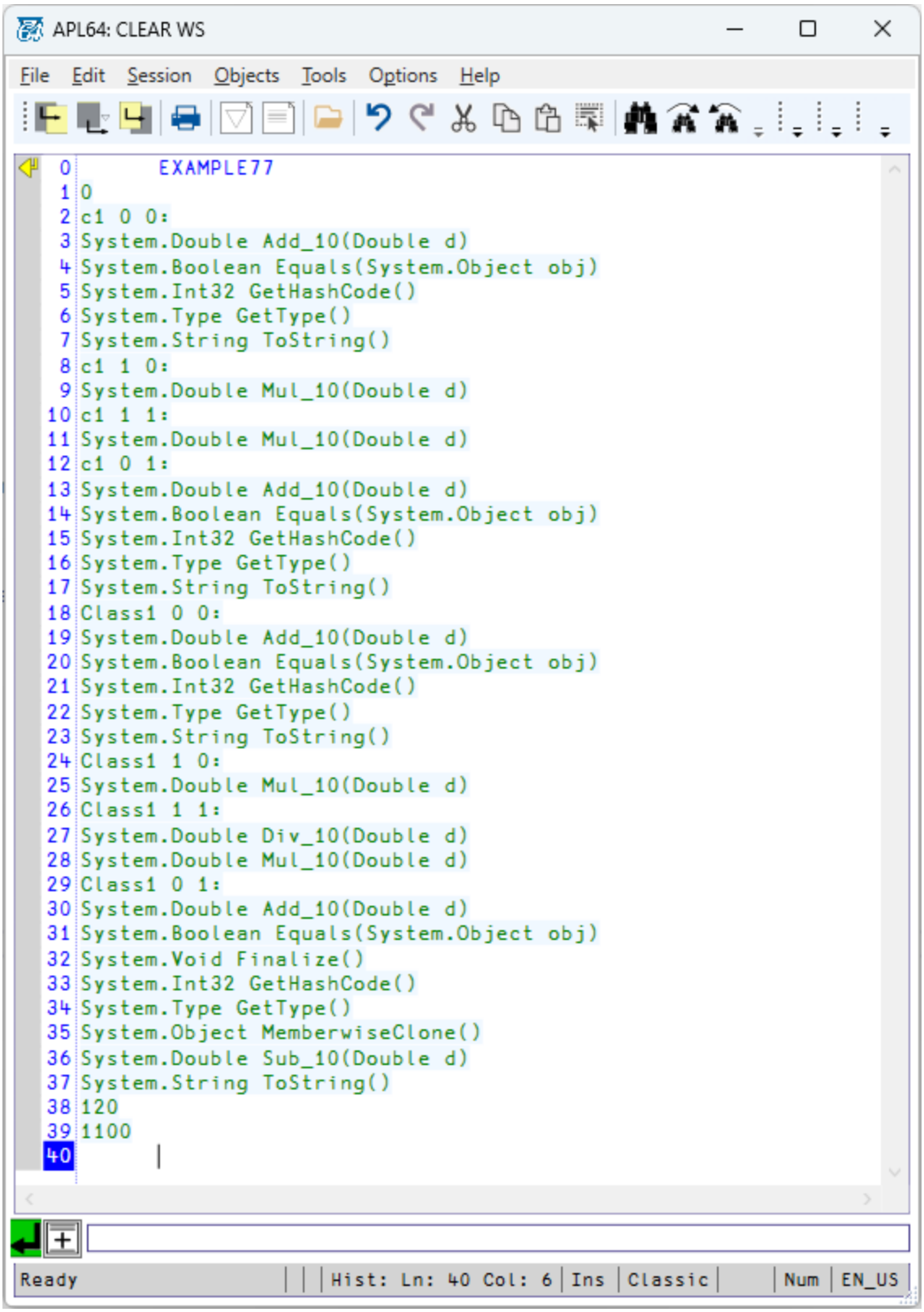

APL64: CLEAR WS
File Edit Session Objects Tools Options Help
0 EXAMPLE76
1 0
2 120
3 100
4 1100
5 11
6 CSE Scope: 0 0:
7 System.Double Add_10(Double d)
8 System.Double Sub_10(Double d)
9 CSE Scope: 1 0:
10 System.Double Div_10(Double d)
11 System.Double Mul_10(Double d)
12 CSE Scope: 1 1:
13 System.Double Div_10(Double d)
14 System.Double Mul_10(Double d)
15 CSE Scope: 0 1:
16 System.Void Finalize()
17 System.Object MemberwiseClone()
18 System.Double Add_10(Double d)
19 System.Void Finalize()
20 System.Object MemberwiseClone()
21 System.Double Sub_10(Double d)
22 System.Void Finalize()
23 System.Object MemberwiseClone()
24 System.Void Finalize()
25 System.Object MemberwiseClone()
26 System.Void Finalize()
27 System.Object MemberwiseClone()
28 System.Void Finalize()
29 System.Object MemberwiseClone()
30 System.Void Finalize()
31 System.Object MemberwiseClone()
32 System.Void Finalize()
33 System.Object MemberwiseClone()
34 System.Void Finalize()
35 System.Object MemberwiseClone()
36 System.Void Finalize()
37 System.Object MemberwiseClone()
38
Ready
Hist: Ln: 38 Col: 6
Ins
Classic
Num
EN_US

Example #77:

In this example an instance class with public and private instance and static methods is illustrated. The results of all variations of the 'GetMethods' method are obtained for a class instance and for the source class.

```
EXAMPLE77;S
S←««
using System;
public class Class1
{
public double Add_10(double d){return 10+d;}
public static double Mul_10(double d){return 10*d;}
private double Sub_10(double d){return d-10;}
private static double Div_10(double d){return d/10;}
}
var c1 = new Class1();
»»
⎕cselF←'cse'⎕cse'Init' 'System'
⎕cse 'Exec' S
'c1 0 0:'
⊃⎕DEB¨⎕cse 'GetMethods' 'c1' 0 0
'c1 1 0:'
⊃⎕DEB¨⎕cse 'GetMethods' 'c1' 1 0
'c1 1 1:'
⊃⎕DEB¨⎕cse 'GetMethods' 'c1' 1 1
'c1 0 1:'
⊃⎕DEB¨⎕cse 'GetMethods' 'c1' 0 1
'Class1 0 0:'
⊃⎕DEB¨⎕cse 'GetMethods' 'Class1' 0 0
'Class1 1 0:'
⊃⎕DEB¨⎕cse 'GetMethods' 'Class1' 1 0
'Class1 1 1:'
⊃⎕DEB¨⎕cse 'GetMethods' 'Class1' 1 1
'Class1 0 1:'
⊃⎕DEB¨⎕cse 'GetMethods' 'Class1' 0 1
⎕cse 'GetValue' 'c1.Add_10(110)'
⎕cse 'GetValue' 'Class1.Mul_10(110)'
⎕cse 'GetLastError'
⎕cse 'Close'
```


∇EXAMPLE77
0 EXAMPLE77;S
1 S←⍬⍬
2 using System;
3 public class Class1
4 {
5 public double Add_10(double d){return 10+d;}
6 public static double Mul_10(double d){return 10*d;}
7 private double Sub_10(double d){return d-10;}
8 private static double Div_10(double d){return d/10;}
9 }
10 var c1 = new Class1();
11 ⍝⍝
12 ⎕cself←'cse'⎕cse'Init' 'System'
13 ⎕cse 'Exec' S
14 'c1 0 0:'
15 ⍎⎕DEB⍠⎕cse 'GetMethods' 'c1' 0 0
16 'c1 1 0:'
17 ⍎⎕DEB⍠⎕cse 'GetMethods' 'c1' 1 0
18 'c1 1 1:'
19 ⍎⎕DEB⍠⎕cse 'GetMethods' 'c1' 1 1
20 'c1 0 1:'
21 ⍎⎕DEB⍠⎕cse 'GetMethods' 'c1' 0 1
22 'Class1 0 0:'
23 ⍎⎕DEB⍠⎕cse 'GetMethods' 'Class1' 0 0
24 'Class1 1 0:'
25 ⍎⎕DEB⍠⎕cse 'GetMethods' 'Class1' 1 0
26 'Class1 1 1:'
27 ⍎⎕DEB⍠⎕cse 'GetMethods' 'Class1' 1 1
28 'Class1 0 1:'
29 ⍎⎕DEB⍠⎕cse 'GetMethods' 'Class1' 0 1
30 ⎕cse 'GetValue' 'c1.Add_10(110)'
31 ⎕cse 'GetValue' 'Class1.Mul_10(110)'
32 ⎕cse 'GetLastError'
33 ⎕cse 'Close'
[33;12]
Commit Changes
Commit & Close


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
0 EXAMPLE77
1 0
2 c1 0 0:
3 System.Double Add_10(Double d)
4 System.Boolean Equals(System.Object obj)
5 System.Int32 GetHashCode()
6 System.Type GetType()
7 System.String ToString()
8 c1 1 0:
9 System.Double Mul_10(Double d)
10 c1 1 1:
11 System.Double Mul_10(Double d)
12 c1 0 1:
13 System.Double Add_10(Double d)
14 System.Boolean Equals(System.Object obj)
15 System.Int32 GetHashCode()
16 System.Type GetType()
17 System.String ToString()
18 Class1 0 0:
19 System.Double Add_10(Double d)
20 System.Boolean Equals(System.Object obj)
21 System.Int32 GetHashCode()
22 System.Type GetType()
23 System.String ToString()
24 Class1 1 0:
25 System.Double Mul_10(Double d)
26 Class1 1 1:
27 System.Double Div_10(Double d)
28 System.Double Mul_10(Double d)
29 Class1 0 1:
30 System.Double Add_10(Double d)
31 System.Boolean Equals(System.Object obj)
32 System.Void Finalize()
33 System.Int32 GetHashCode()
34 System.Type GetType()
35 System.Object MemberwiseClone()
36 System.Double Sub_10(Double d)
37 System.String ToString()
38 120
39 1100
40
Ready
Hist: Ln: 40 Col: 6
Ins
Classic
Num
EN_US

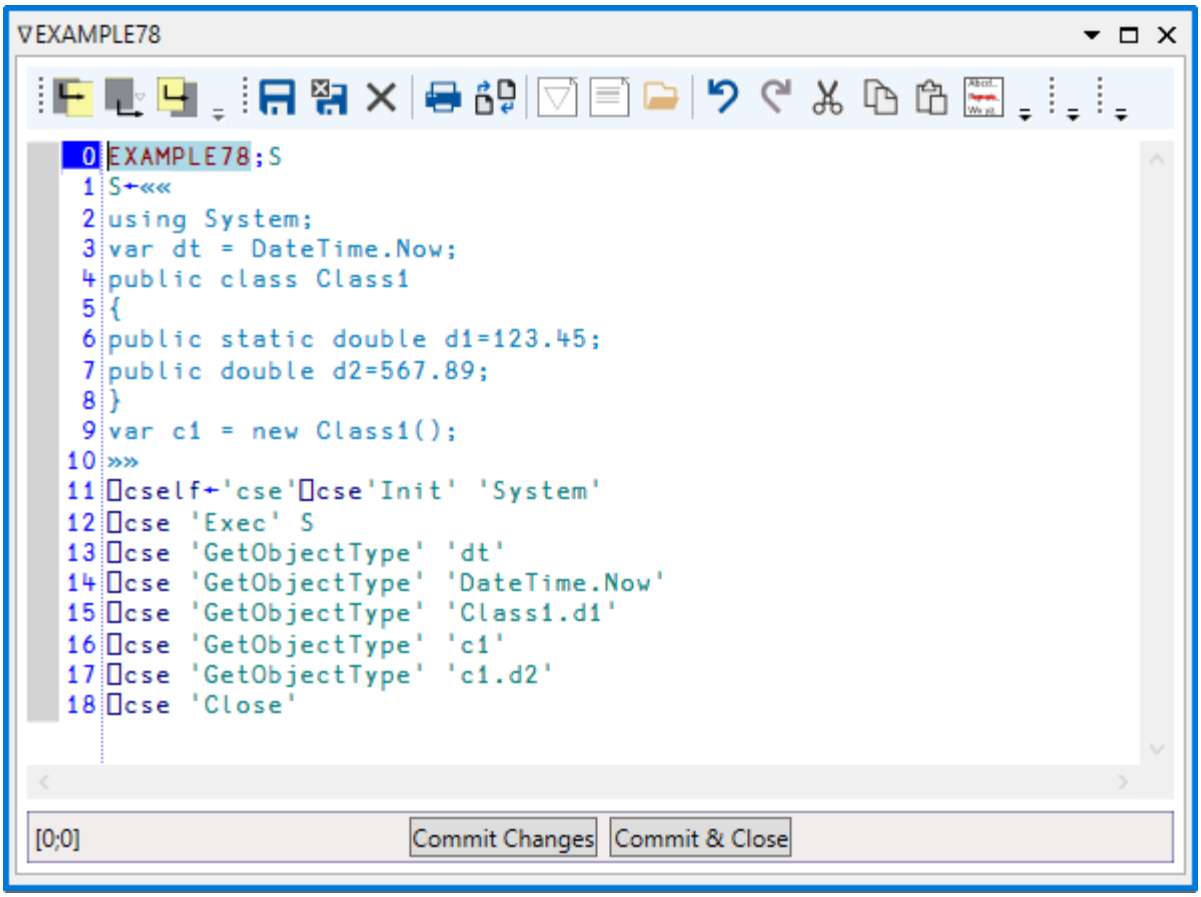
CSE GetObjectType Method

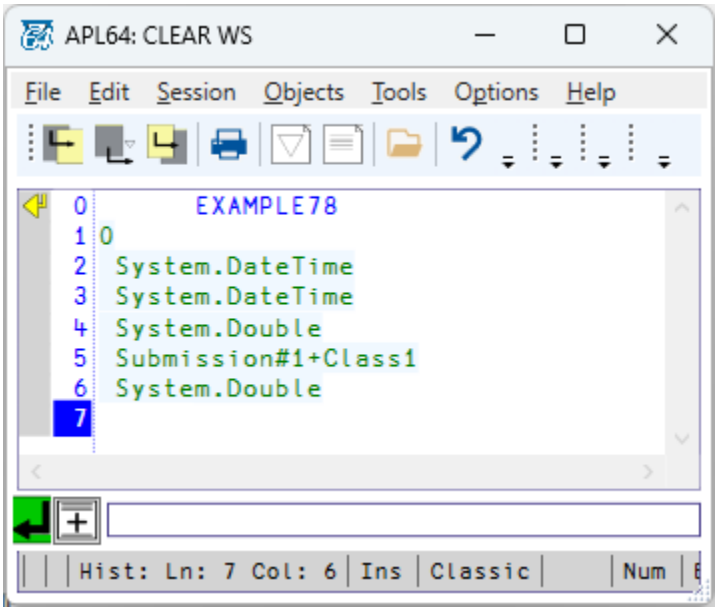
The CSE 'GetObjectType' method returns a text vector containing the name of the .Net data type of a public .Net object.

Example #78:

'dt' is the name of an instance of DateTime. 'Class1.d1' is a static field in Class1. 'c1' is an instance of Class1. 'c1.d2' is an instance field of 'c1'.

```
EXAMPLE78;S
S←««
using System;
var dt = DateTime.Now;
public class Class1
{
public static double d1=123.45;
public double d2=567.89;
}
var c1 = new Class1();
»»
□cselⅈ←'cse'□cse'Init' 'System'
□cse 'Exec' S
□cse 'GetObjectType' 'dt'
□cse 'GetObjectType' 'DateTime.Now'
□cse 'GetObjectType' 'Class1.d1'
□cse 'GetObjectType' 'c1'
□cse 'GetObjectType' 'c1.d2'
□cse 'Close'
```





CSE GetProperties Method

The CSE 'GetProperties' method returns a vector of text vectors containing the properties of the .Net object specified in the right argument of the CSE 'GetProperties' method. The results are sorted

alphabetically in increasing order according to the .Net object's property name. Each element of the result of the CSE 'GetProperties' method indicates:

- The name of the .Net property
- The .Net type of the property value as a prefix to the property name
- If the property provides a 'get' and/or 'set' method as a suffix to the property name

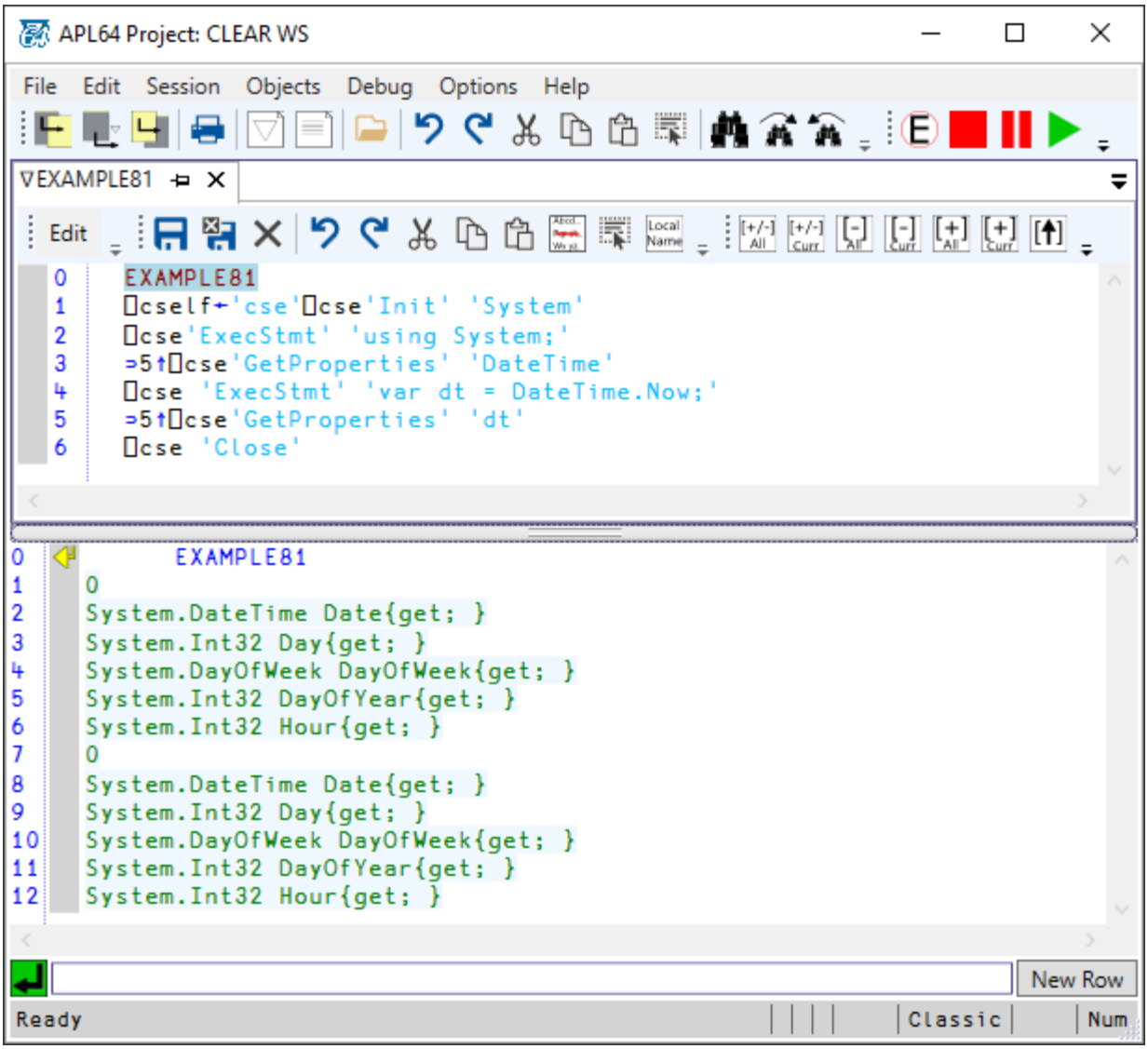
Right Arg#	**Arg Description**
1	Required: Name of class or instance of class or '' for properties defined within the scope of a CSE instance
2	Optional: 0/Return Instance Properties(default) 1/Return Static Properties
3	Optional: 0/Return Public Properties(default) 1/Return Public & Private Properties

Required Right Argument of the CSE 'GetProperties' method:

- For public properties defined within the scope of a .Net class.
    - The 1st right argument of the CSE 'GetProperties' method should be the name of an instance the class if the class is an instance class
    - The 1st right argument of the CSE 'GetProperties' methods should be the name of the class if the class is a static class or if the instance class has static properties.
- For properties defined within the scope of a CSE instance, the right argument of the CSE
- 'GetProperties' method is '' (i.e. 0ρ' ').

Example #81:
This example illustrates the static properties of the DateTime class and the properties of an instance of the dateTime class, 'dt'.

```
EXAMPLE81
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse'ExecStmt' 'using System;'
⊃5↑⎕cse'GetProperties' 'DateTime'
⎕cse 'ExecStmt' 'var dt = DateTime.Now;'
⊃5↑⎕cse'GetProperties' 'dt'
⎕cse 'Close'
```

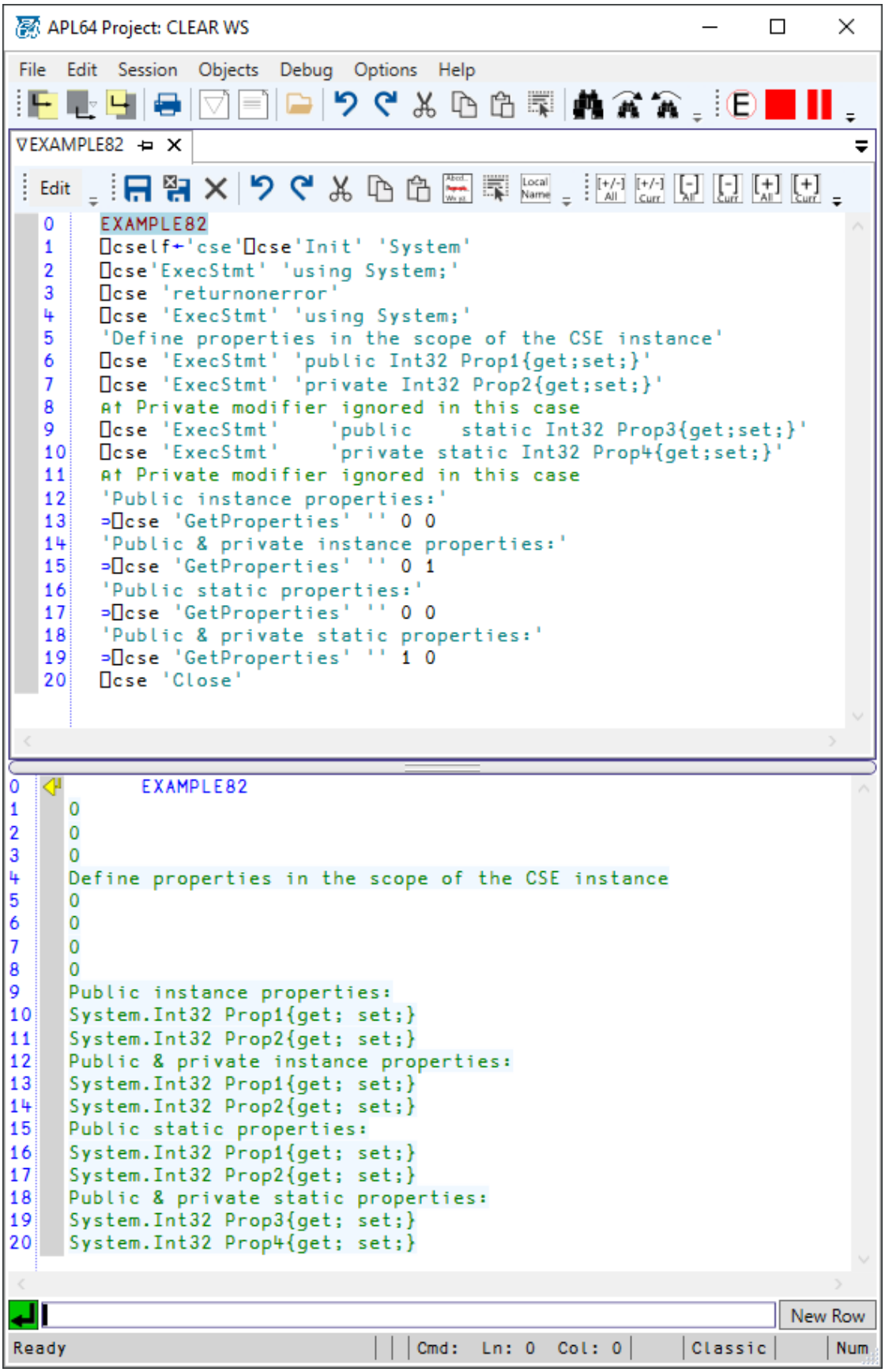


Example #82:

This example illustrates properties defined in the scope of the CSE instance. For properties which are defined in the scope of the CSE instance, the inclusion of a 'private' modified is ignored, because such properties are defined at the 'root level' of a CSE instance and if they were private would not be accessible. Thus, the 3rd right argument of the CSE 'GetProperties' method has no effect in this example.

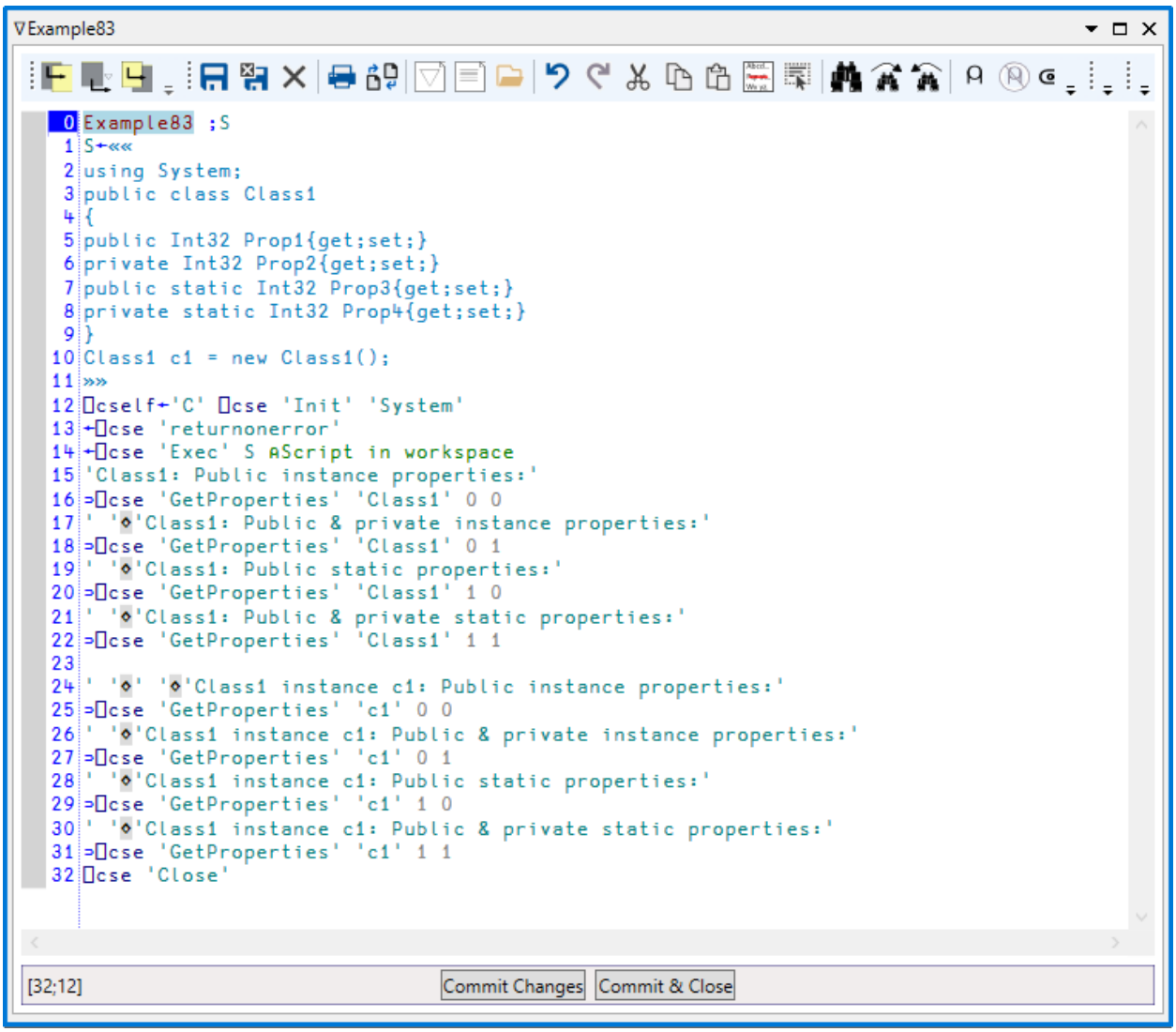
```
EXAMPLE82
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse'ExecStmt' 'using System;'
⎕cse 'returnonerror'
⎕cse 'ExecStmt' 'using System;'
'Define properties in the scope of the CSE instance'
⎕cse 'ExecStmt' 'public Int32 Prop1{get;set;}'
⎕cse 'ExecStmt' 'private Int32 Prop2{get;set;}'
```

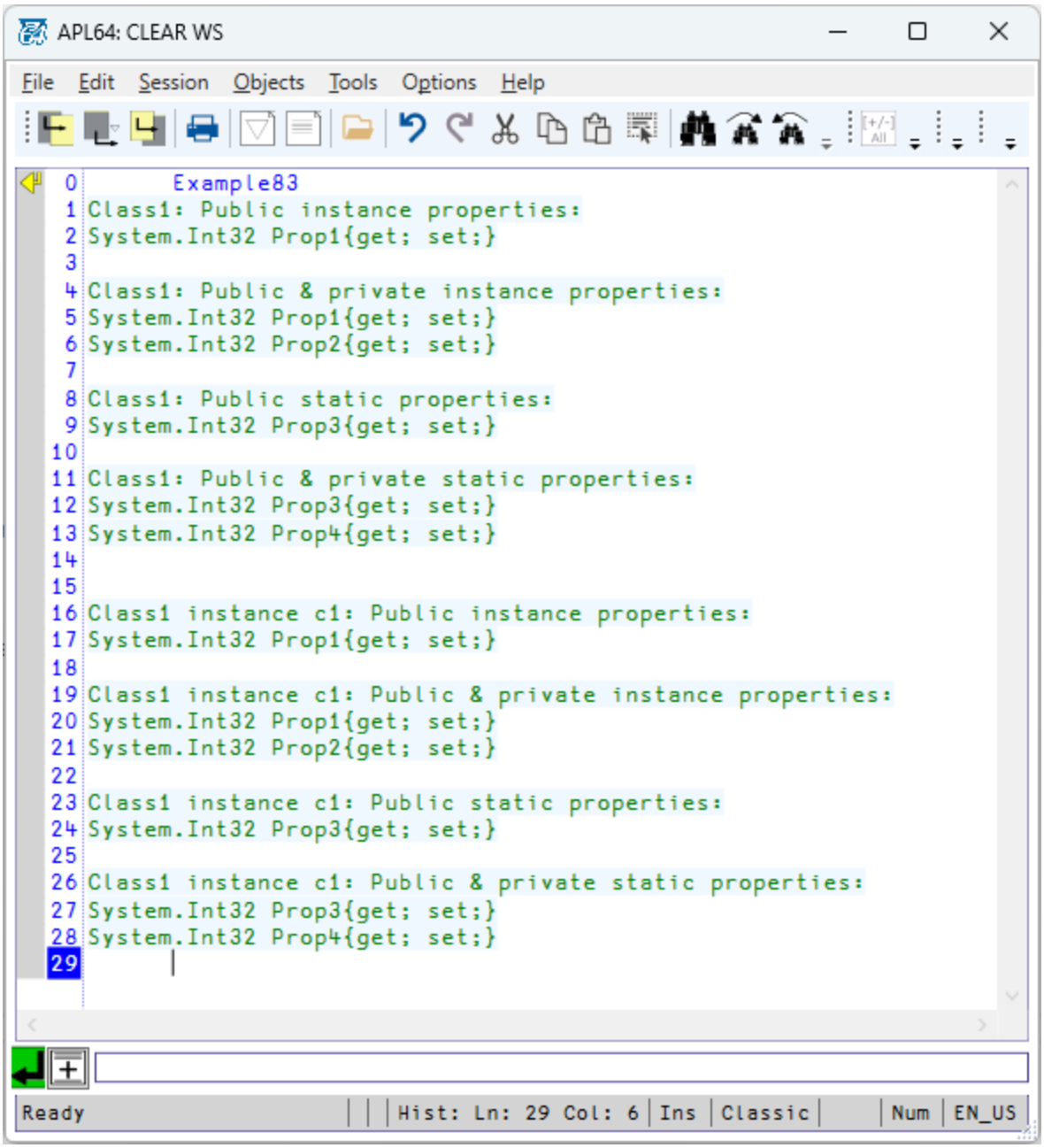
```
⍝↑ Private modifier ignored in this case
⎕cse 'ExecStmt'    'public    static Int32 Prop3{get;set;}'
⎕cse 'ExecStmt'    'private static Int32 Prop4{get;set;}'
⍝↑ Private modifier ignored in this case
'Public instance properties:'
⊃⎕cse 'GetProperties' '' 0 0
'Public & private instance properties:'
⊃⎕cse 'GetProperties' '' 0 1
'Public static properties:'
⊃⎕cse 'GetProperties' '' 0 0
'Public & private static properties:'
⊃⎕cse 'GetProperties' '' 1 0
⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
∇EXAMPLE82
Edit
0 EXAMPLE82
1 ⎕cself←'cse'⎕cse'Init' 'System'
2 ⎕cse'ExecStmt' 'using System;'
3 ⎕cse 'returnonerror'
4 ⎕cse 'ExecStmt' 'using System;'
5 'Define properties in the scope of the CSE instance'
6 ⎕cse 'ExecStmt' 'public Int32 Prop1{get;set;}'
7 ⎕cse 'ExecStmt' 'private Int32 Prop2{get;set;}'
8 ⍝ Private modifier ignored in this case
9 ⎕cse 'ExecStmt' 'public static Int32 Prop3{get;set;}'
10 ⎕cse 'ExecStmt' 'private static Int32 Prop4{get;set;}'
11 ⍝ Private modifier ignored in this case
12 'Public instance properties:'
13 ⍎⎕cse 'GetProperties' '' 0 0
14 'Public & private instance properties:'
15 ⍎⎕cse 'GetProperties' '' 0 1
16 'Public static properties:'
17 ⍎⎕cse 'GetProperties' '' 0 0
18 'Public & private static properties:'
19 ⍎⎕cse 'GetProperties' '' 1 0
20 ⎕cse 'Close'
0 EXAMPLE82
1 0
2 0
3 0
4 Define properties in the scope of the CSE instance
5 0
6 0
7 0
8 0
9 Public instance properties:
10 System.Int32 Prop1{get; set;}
11 System.Int32 Prop2{get; set;}
12 Public & private instance properties:
13 System.Int32 Prop1{get; set;}
14 System.Int32 Prop2{get; set;}
15 Public static properties:
16 System.Int32 Prop1{get; set;}
17 System.Int32 Prop2{get; set;}
18 Public & private static properties:
19 System.Int32 Prop3{get; set;}
20 System.Int32 Prop4{get; set;}
New Row
Ready
Cmd: Ln: 0 Col: 0
Classic
Num

Example #83:
This example illustrates properties defined in a custom class. The CSE ‘GetProperties’ method reports only public properties.

```
Example83 ;S
S←««
using System;
public class Class1
{
public Int32 Prop1{get;set;}
private Int32 Prop2{get;set;}
public static Int32 Prop3{get;set;}
private static Int32 Prop4{get;set;}
}
Class1 c1 = new Class1();
»»
□cself←'C' □cse 'Init' 'System'
←□cse 'returnonerror'
←□cse 'Exec' S ⍝Script in workspace
'Class1: Public instance properties:'
⊃□cse 'GetProperties' 'Class1' 0 0
' '⋄'Class1: Public & private instance properties:'
⊃□cse 'GetProperties' 'Class1' 0 1
' '⋄'Class1: Public static properties:'
⊃□cse 'GetProperties' 'Class1' 1 0
' '⋄'Class1: Public & private static properties:'
⊃□cse 'GetProperties' 'Class1' 1 1

' '⋄' '⋄'Class1 instance c1: Public instance properties:'
⊃□cse 'GetProperties' 'c1' 0 0
' '⋄'Class1 instance c1: Public & private instance properties:'
⊃□cse 'GetProperties' 'c1' 0 1
' '⋄'Class1 instance c1: Public static properties:'
⊃□cse 'GetProperties' 'c1' 1 0
' '⋄'Class1 instance c1: Public & private static properties:'
⊃□cse 'GetProperties' 'c1' 1 1
□cse 'Close'
```


Example83
0 Example83 ;S
1 S←⍬⍬
2 using System;
3 public class Class1
4 {
5 public Int32 Prop1{get;set;}
6 private Int32 Prop2{get;set;}
7 public static Int32 Prop3{get;set;}
8 private static Int32 Prop4{get;set;}
9 }
10 Class1 c1 = new Class1();
11 ⍬⍬
12 ⎕cself←'C' ⎕cse 'Init' 'System'
13 ←⎕cse 'returnonerror'
14 ←⎕cse 'Exec' S ⍝Script in workspace
15 'Class1: Public instance properties:'
16 ⎕cse 'GetProperties' 'Class1' 0 0
17 ' '⋄'Class1: Public & private instance properties:'
18 ⎕cse 'GetProperties' 'Class1' 0 1
19 ' '⋄'Class1: Public static properties:'
20 ⎕cse 'GetProperties' 'Class1' 1 0
21 ' '⋄'Class1: Public & private static properties:'
22 ⎕cse 'GetProperties' 'Class1' 1 1
23
24 ' '⋄' '⋄'Class1 instance c1: Public instance properties:'
25 ⎕cse 'GetProperties' 'c1' 0 0
26 ' '⋄'Class1 instance c1: Public & private instance properties:'
27 ⎕cse 'GetProperties' 'c1' 0 1
28 ' '⋄'Class1 instance c1: Public static properties:'
29 ⎕cse 'GetProperties' 'c1' 1 0
30 ' '⋄'Class1 instance c1: Public & private static properties:'
31 ⎕cse 'GetProperties' 'c1' 1 1
32 ⎕cse 'Close'
[32;12]
Commit Changes
Commit & Close



CSE GetReferences Method

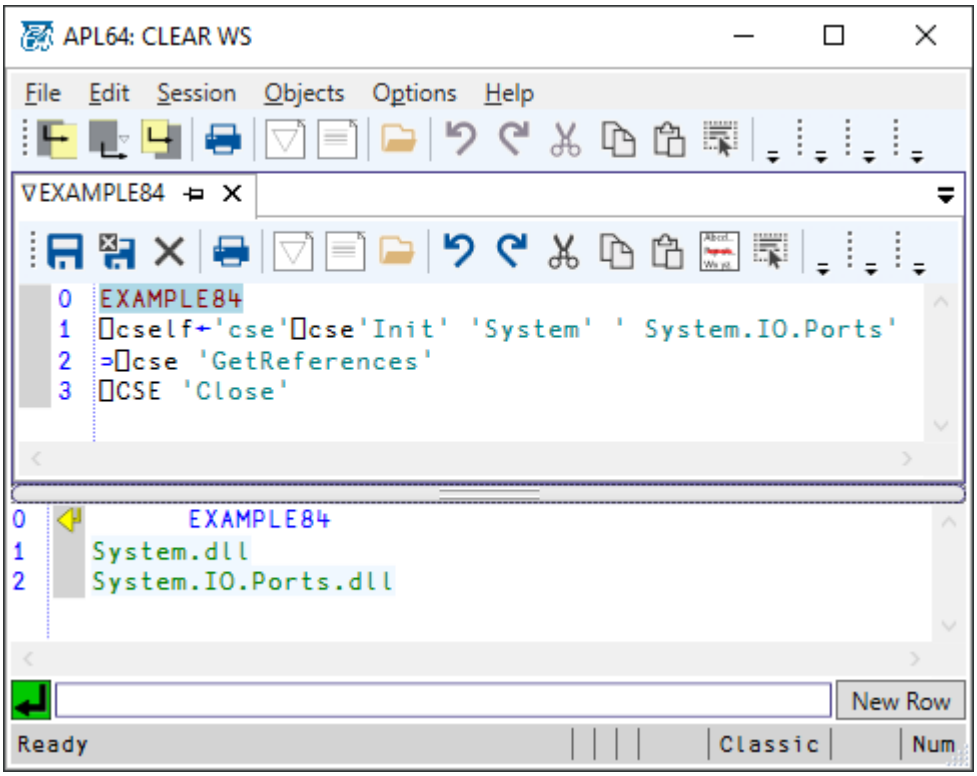
The CSE 'GetReferences' method returns a vector of text vectors of the names of the .Net assemblies loaded into the instance of the CSE object. There is no right argument to this method.

The CSE 'GetReferences' method will report .Net assemblies which are loaded using the CSE 'Init', 'LoadAssembly' and 'LoadAssemblyByName' methods.

Example #84: Two assemblies have been loaded into a CSE instance: System and System.IO.Ports.

```
EXAMPLE84
```

```
⎕cself←'cse'⎕cse'Init' 'System' ' System.IO.Ports'
⊃⎕cse 'GetReferences'
⎕CSE 'Close'
```



CSE GetRoutedEvents Method

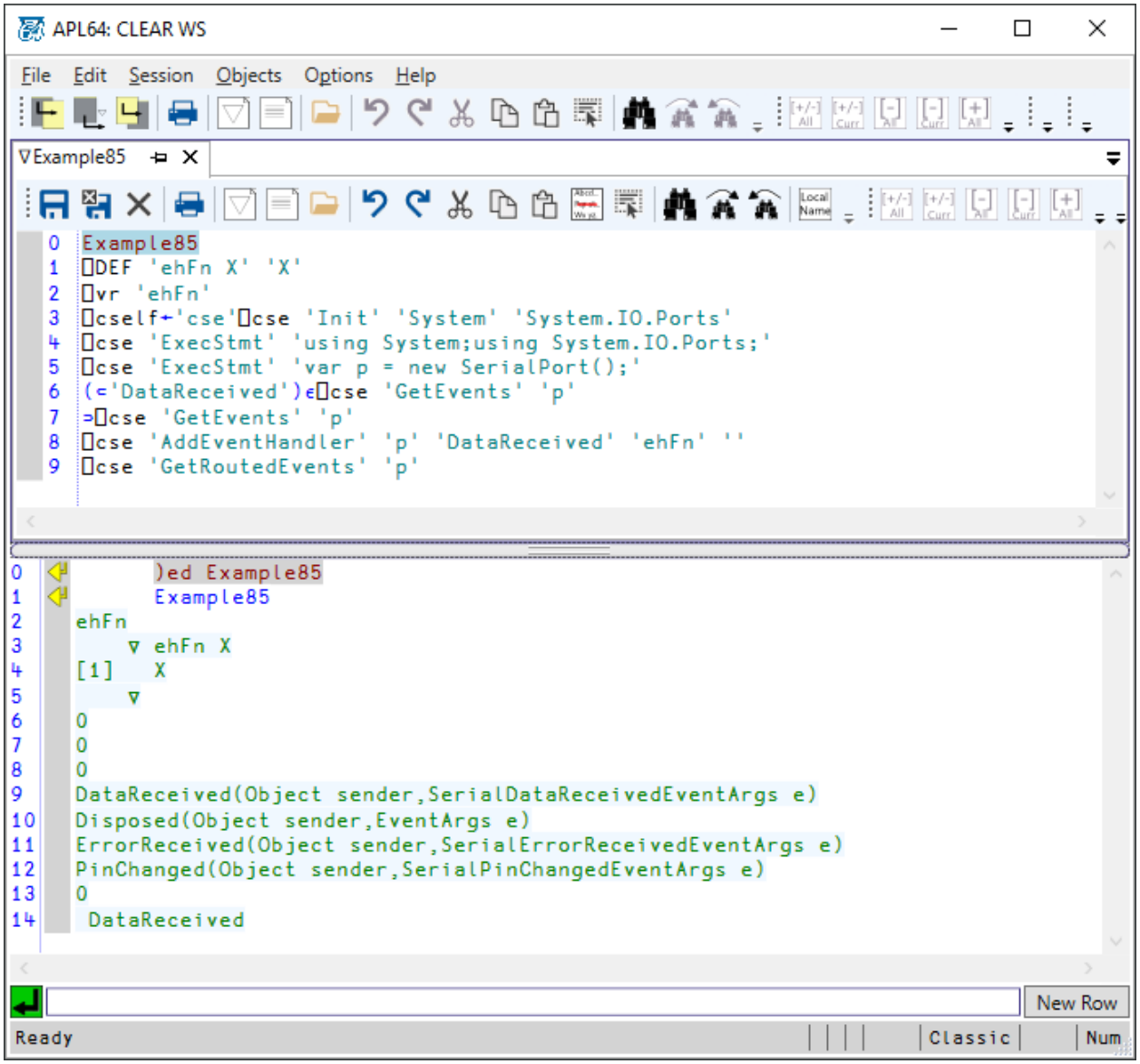
The CSE 'GetRoutedEvents' method returns a vector of text vectors containing the names of the routed events exposed by a the C# object specified in the right argument to the CSE 'GetRoutedEvents' method which have been subscribed using the CSE 'AddEventHandler' or 'AddEventHandlerEx' methods. The CSE object specified in the right argument of the CSE 'GetRoutedEvents' must exist in the state of the CSE object.

Example #85:

In this example an instance of the .Net System.IO.Ports.Port object type is created. The CSE 'AddEventHandler' method subscribes to the .Net 'HelpButtonClicked' event of the .Net Form object with the APL64 event handler function 'EHfn'. After the event subscription is completed, the CSE 'GetRoutedEvents' method is used to see that the .Net 'HelpButtonClicked' event has been subscribed by this instance of the CSE object, 'fm'.

```
Example85
⎕DEF 'ehFn X' 'X'
⎕vr 'ehFn'
⎕cself←'cse'⎕cse 'Init' 'System' 'System.IO.Ports'
```

```
⎕cse 'ExecStmt' 'using System;using System.IO.Ports;'
⎕cse 'ExecStmt' 'var p = new SerialPort();'
(⊂'DataReceived')∈⎕cse 'GetEvents' 'p'
⊃⎕cse 'GetEvents' 'p'
⎕cse 'AddEventHandler' 'p' 'DataReceived' 'ehFn' ''
⎕cse 'GetRoutedEvents' 'p'
```



CSE GetSessionTypes Method

The CSE 'GetSessionTypes' returns a vector of text vectors with the names of the .Net types which have been defined in the specified instance of the CSE, for example definitions of instance or static classes, structures or enumerations. The hierarchy of type structure is included in the result where each level delimited by a '+' sign. By design, the names of types inherently defined in .Net assemblies loaded into the current CSE instance are not returned by the CSE 'GetSessionTypes' method. The CSE

'GetSessionTypes' method has no right argument. The names of .Net value types, e.g. Int32, double, etc. and strings, are not returned by the CSE 'GetSessionTypes' method, but the names of most .Net reference types, e.g. classes, arrays of value types, etc. are returned by this CSE method.

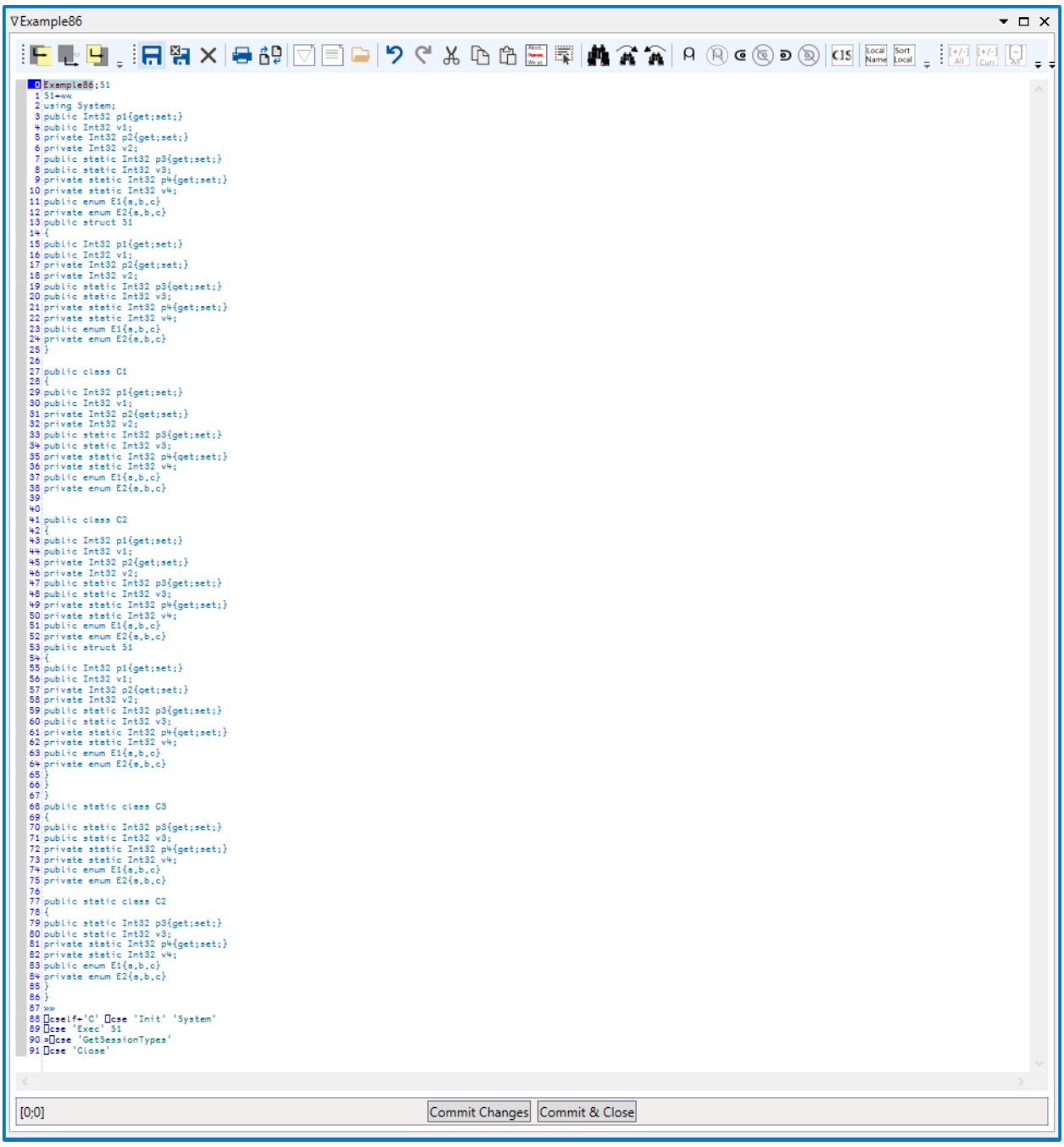
Some .Net type names returned by this method may be those created by the CSE itself.

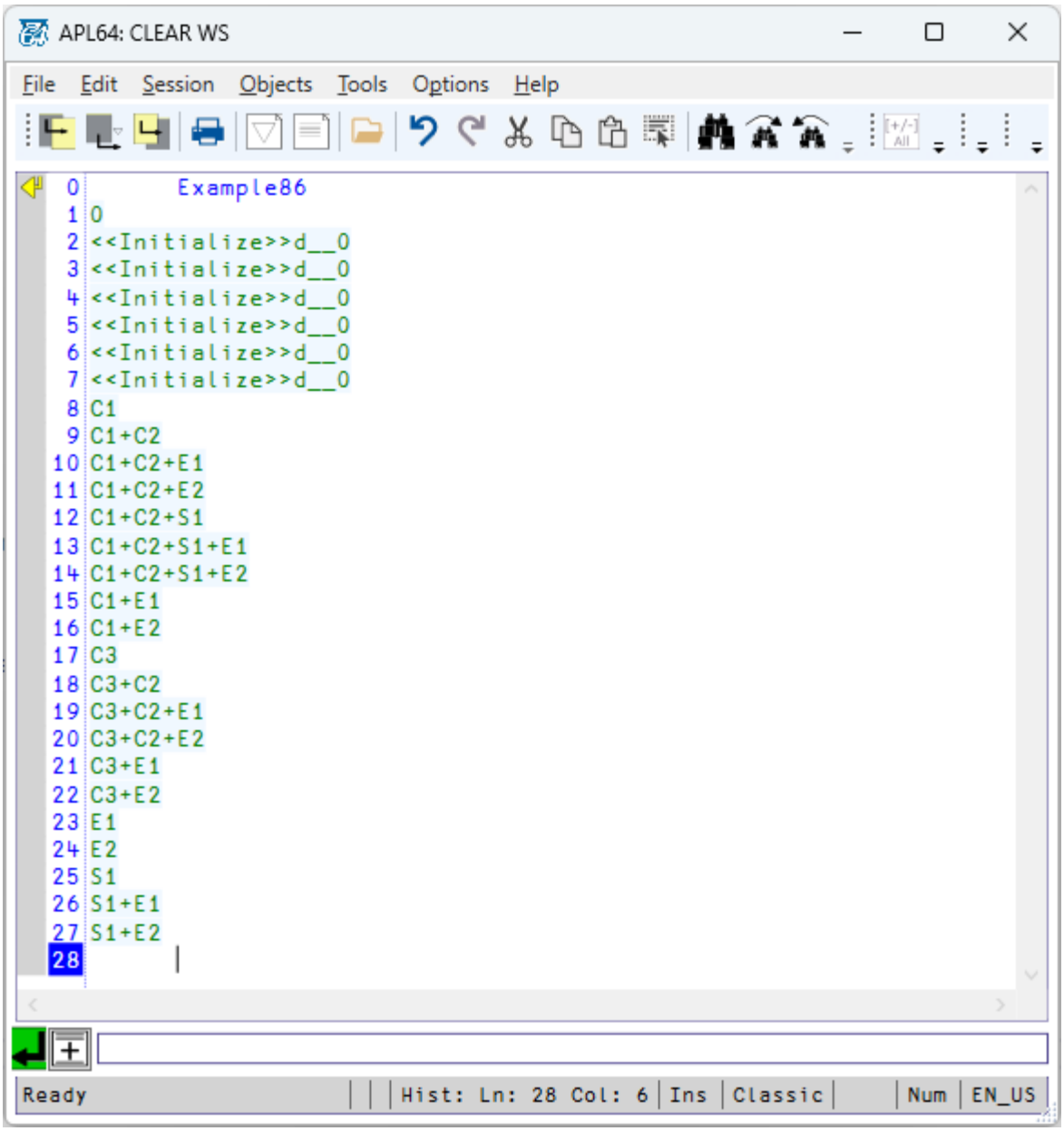
Example #86:
In this example public, private, instance and static types have been defined by the script 'S'. For example the CSE 'GetSessionTypes' result element 'C1+C2+S1+E1' indicates that in the current CSE session there is a definition of the type E1 (enumeration) which is defined in S1 (structure) which is defined in C2 (class) (nested class) defined in C1 (class).

```
Example86;S1
S1←««
using System;
public Int32 p1{get;set;}
public Int32 v1;
private Int32 p2{get;set;}
private Int32 v2;
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}
public struct S1
{
public Int32 p1{get;set;}
public Int32 v1;
private Int32 p2{get;set;}
private Int32 v2;
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}
}

public class C1
{
public Int32 p1{get;set;}
```

```
public Int32 v1;
private Int32 p2{get;set;}
private Int32 v2;
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}


public class C2
{
public Int32 p1{get;set;}
public Int32 v1;
private Int32 p2{get;set;}
private Int32 v2;
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}
public struct S1
{
public Int32 p1{get;set;}
public Int32 v1;
private Int32 p2{get;set;}
private Int32 v2;
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}
}
}
}
public static class C3
{
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
```

```
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}

public static class C2
{
public static Int32 p3{get;set;}
public static Int32 v3;
private static Int32 p4{get;set;}
private static Int32 v4;
public enum E1{a,b,c}
private enum E2{a,b,c}
}
}
»»
□cself←'C' □cse 'Init' 'System'
□cse 'Exec' S1
⊃□cse 'GetSessionTypes'
□cse 'Close'
```


Example86
Commit Changes
Commit & Close
[0;0]


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
0 Example86
1 0
2 <<Initialize>>d__0
3 <<Initialize>>d__0
4 <<Initialize>>d__0
5 <<Initialize>>d__0
6 <<Initialize>>d__0
7 <<Initialize>>d__0
8 C1
9 C1+C2
10 C1+C2+E1
11 C1+C2+E2
12 C1+C2+S1
13 C1+C2+S1+E1
14 C1+C2+S1+E2
15 C1+E1
16 C1+E2
17 C3
18 C3+C2
19 C3+C2+E1
20 C3+C2+E2
21 C3+E1
22 C3+E2
23 E1
24 E2
25 S1
26 S1+E1
27 S1+E2
28
Ready
Hist: Ln: 28 Col: 6
Ins
Classic
Num
EN_US

CSE GetValue Method

The 'GetValue' method will assign the value of a C# source object property or method result to the value of an APL64 target object. The 'GetValue' method has one required right argument that is a text vector containing the name of the pre-existing C# source object property or method and the method's argument, if any.

The data type of the source C# object property or method result must be conformable with the data type of the target APL64 object, or an APL64 exception will be thrown. APL has inherent data types corresponding to the .Net value types Int32, double and string as well as arrays of those value types. The 'GetValue' method can be used to directly obtain in APL64 the value of .Net properties or method results with such value types.

More complex .Net objects in C# have an object model which is a hierarchy of other .Net objects which ultimately reduce to .Net value types with analogous APL64 data types. For example, the .Net DateTime data type has Year, Month and Day properties which are Int32 data types. Because the data types of these subordinate properties have APL64 analogues, the values of each of these properties can be assigned to APL variables using the CSE 'GetValue' method.

If the CSE 'GetValue' method is used to obtain the value of a C# variable of a more complex type than that supported by APL, the results may need further interpretation or the method may throw an APL64 exception. For example, the result of 'GetValue' applied to a C# DateTime object would need to be converted from the Microsoft date number to an appropriate APL64 value.

Because there is no way to distinguish a valid returned value from an error code, the CSE instance 'returnonerror' property cannot apply to the CSE GetValue method. For the processing of the .Net value returned by the GetValue method, the effective value of the CSE 'returnonerror' property is 1 and CSE exceptions will be thrown if the returned data type has no representation in APL64.

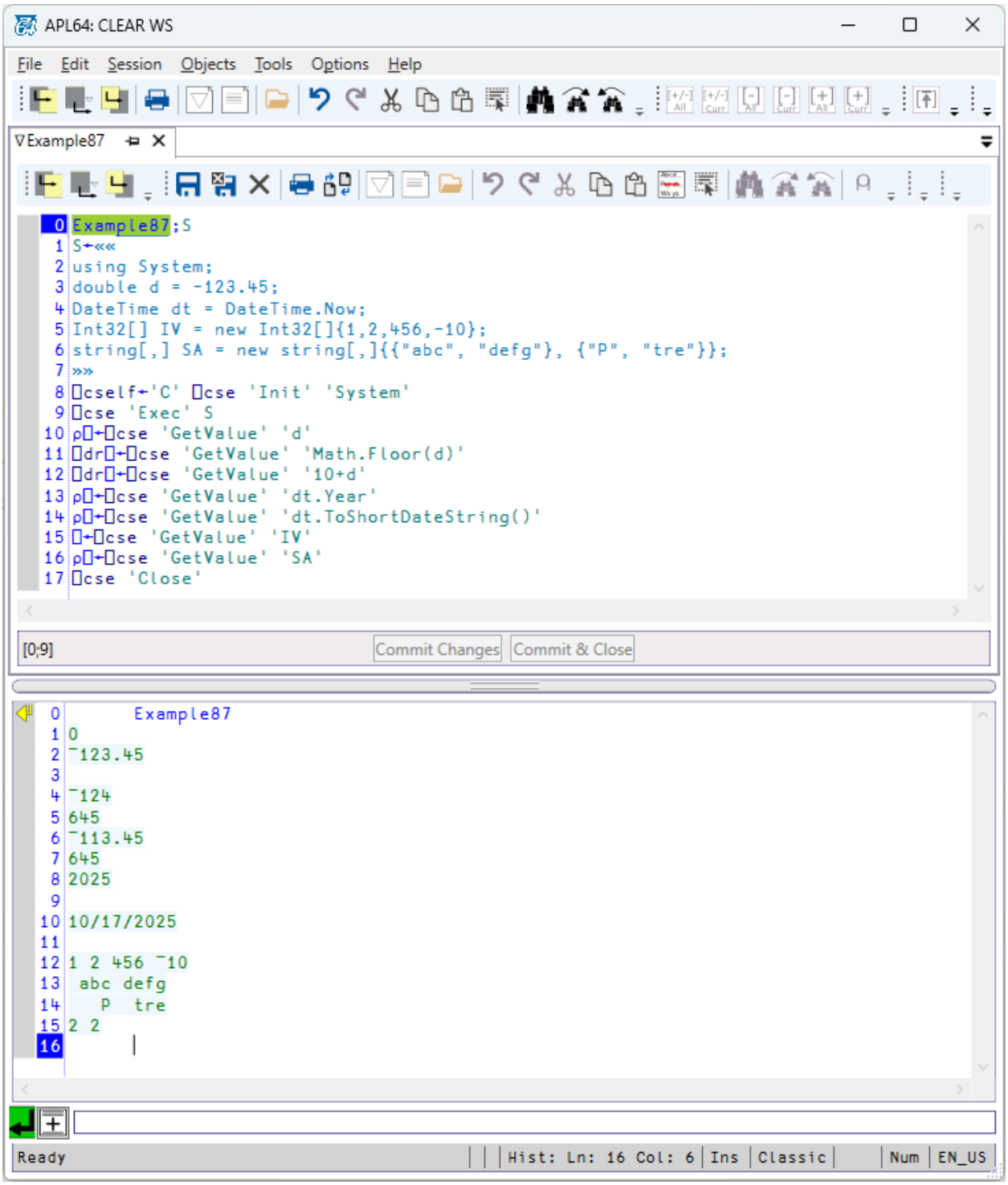
The CSE 'GetValue' method supports value substitution of APL64 values.

Example #87:

In this example .Net DateTime, double, Int32[] and string[,] variables are defined. Their values and the values of the results of some methods applicable to these variables are obtained in APL64 using the CSE 'GetValue' method:.

```
Example87;S
S←««
using System;
double d = -123.45;
```

```
DateTime dt = DateTime.Now;
Int32[] IV = new Int32[]{1,2,456,-10};
string[,] SA = new string[,]{{"abc", "defg"}, {"P", "tre"}};
»»
□cself←'C' □cse 'Init' 'System'
□cse 'Exec' S
ρ□←□cse 'GetValue' 'd'
□dr□←□cse 'GetValue' 'Math.Floor(d)'
□dr□←□cse 'GetValue' '10+d'
ρ□←□cse 'GetValue' 'dt.Year'
ρ□←□cse 'GetValue' 'dt.ToShortDateString()'
□←□cse 'GetValue' 'IV'
ρ□←□cse 'GetValue' 'SA'
□cse 'Close'
```

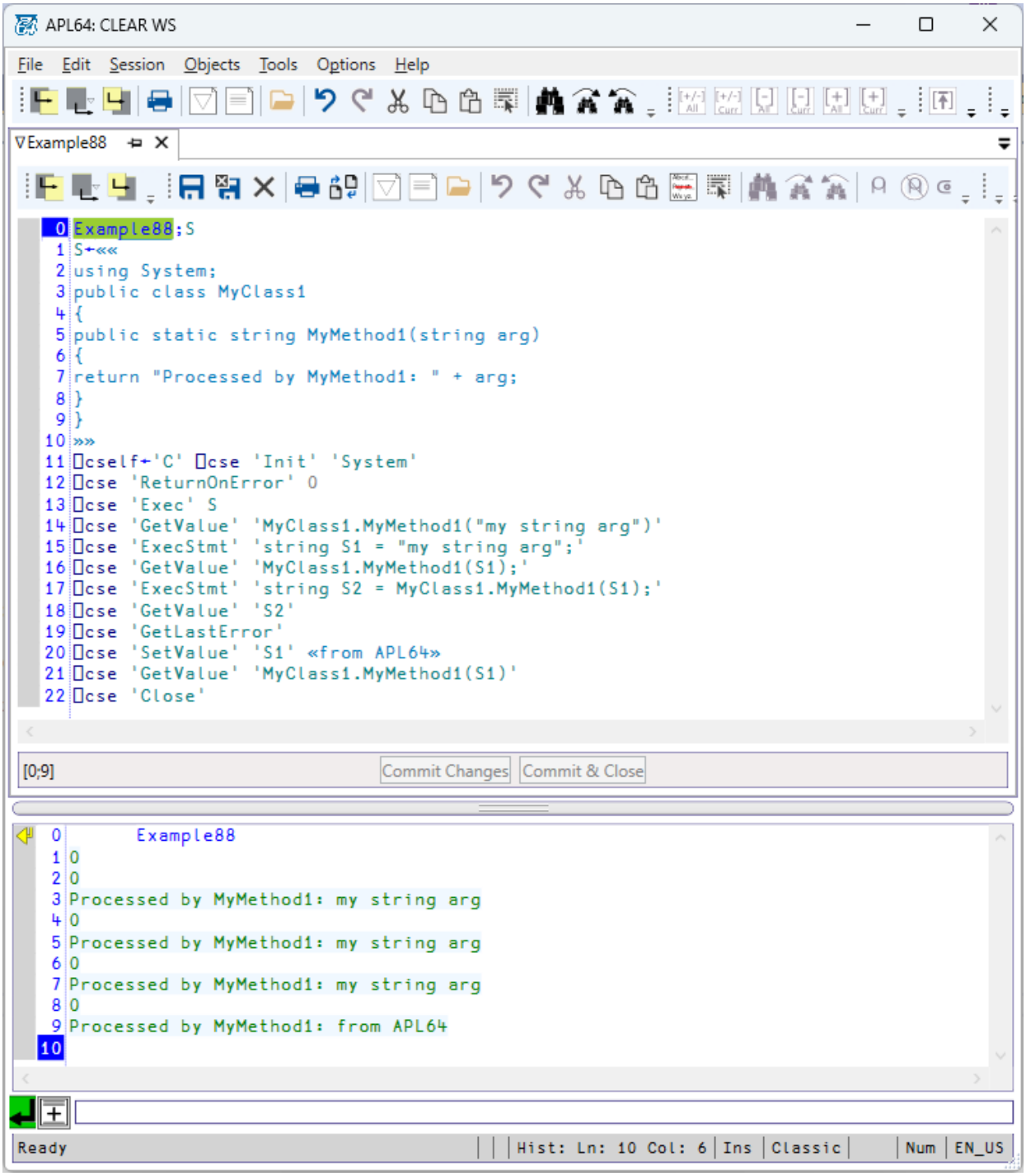


Example #88:

This example illustrates that the CSE 'GetValue' method can be used to obtain the value of the result of a C# method, defined in a custom class, with a specified argument.

```
Example88;S
S←««
```

```
using System;
public class MyClass1
{
public static string MyMethod1(string arg)
{
return "Processed by MyMethod1: " + arg;
}
}
»»
⍞cself←'C' ⍞cse 'Init' 'System'
⍞cse 'ReturnOnError' 0
⍞cse 'Exec' S
⍞cse 'GetValue' 'MyClass1.MyMethod1("my string arg")'
⍞cse 'ExecStmt' 'string S1 = "my string arg";'
⍞cse 'GetValue' 'MyClass1.MyMethod1(S1);'
⍞cse 'ExecStmt' 'string S2 = MyClass1.MyMethod1(S1);'
⍞cse 'GetValue' 'S2'
⍞cse 'GetLastError'
⍞cse 'SetValue' 'S1' «from APL64»
⍞cse 'GetValue' 'MyClass1.MyMethod1(S1)'
⍞cse 'Close'
```



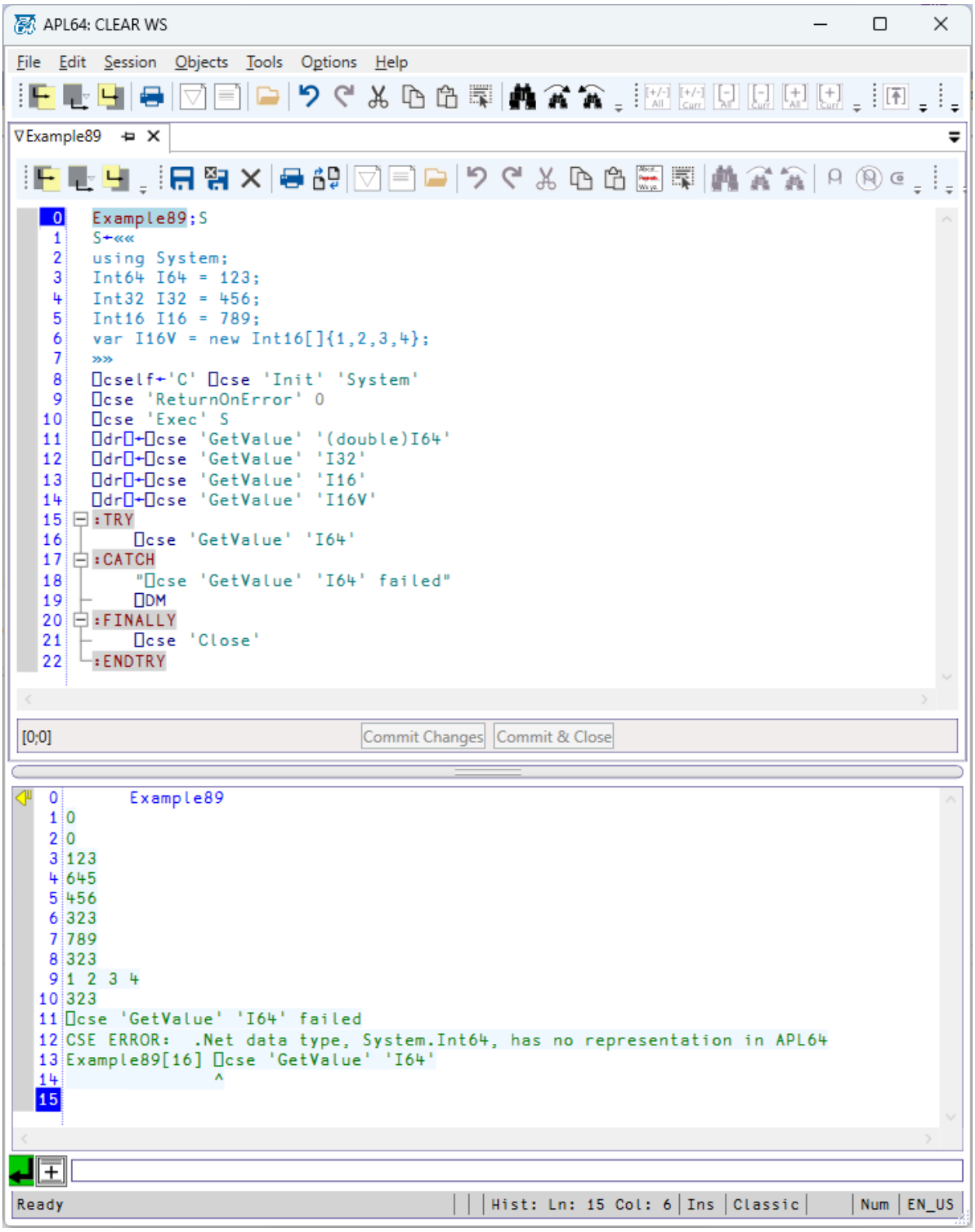
Example #89:
In this example the CSE 'GetValue' method is used to attempt to obtain the value of C# variables with types Int64, Int32 and Int16. The APL64 integer data type is analogous to the .Net Int32 value type.

APL64 does not have a data type which is conformable, without potential loss of precision, with the .Net Int64 data type, so an APL64 exception is thrown and handled by APL64 try/catch/finally/endtry control

structures. By using an explicit C# cast of Int64 value to a double, the APL64 programmer can view the C# value, possibly with a loss of precision.

Since there is no possibility of loss of precision from Int16 to Int32, the CSE implicitly coerces the Int16 value to the APL64 integer (Int32)data type.

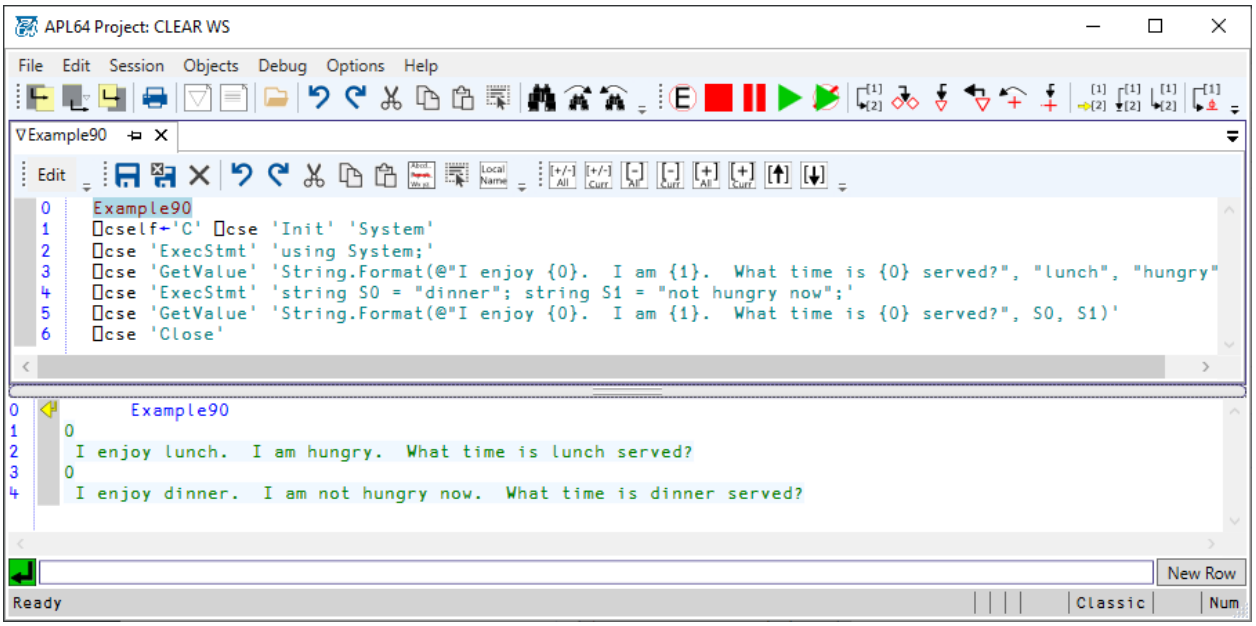
```
Example89;S
S←««
using System;
Int64 I64 = 123;
Int32 I32 = 456;
Int16 I16 = 789;
var I16V = new Int16[]{1,2,3,4};
»»
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'Exec' S
⎕dr⎕←⎕cse 'GetValue' '(double)I64'
⎕dr⎕←⎕cse 'GetValue' 'I32'
⎕dr⎕←⎕cse 'GetValue' 'I16'
⎕dr⎕←⎕cse 'GetValue' 'I16V'
:TRY
   ⎕cse 'GetValue' 'I64'
:CATCH
   "⎕cse 'GetValue' 'I64' failed"
   ⎕DM
:FINALLY
   ⎕cse 'Close'
:ENDTRY
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Example89
Example89;S
S←⍬⍬
using System;
Int64 I64 = 123;
Int32 I32 = 456;
Int16 I16 = 789;
var I16V = new Int16[]{1,2,3,4};
⍬⍬⍬
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'Exec' S
⎕dr⎕←⎕cse 'GetValue' '(double)I64'
⎕dr⎕←⎕cse 'GetValue' 'I32'
⎕dr⎕←⎕cse 'GetValue' 'I16'
⎕dr⎕←⎕cse 'GetValue' 'I16V'
:TRY
⎕cse 'GetValue' 'I64'
:CATCH
"⎕cse 'GetValue' 'I64' failed"
⎕DM
:FINALLY
⎕cse 'Close'
:ENDTRY
[0;0]
Commit Changes
Commit & Close
Example89
0
0
123
645
456
323
789
323
1 2 3 4
323
⎕cse 'GetValue' 'I64' failed
CSE ERROR: .Net data type, System.Int64, has no representation in APL64
Example89[16] ⎕cse 'GetValue' 'I64'
^
Ready
Hist: Ln: 15 Col: 6
Ins
Classic
Num
EN_US

Example #90

In this example the CSE ‘GetValue’ method is used to execute a C# method, ‘String.Format()’, using explicit arguments provided in the C# executable statement and alternatively using C# field values which have been established using the CSE ‘ExecStmt’ method:

```
Example90
 cself←'C'  cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'
 cse 'GetValue' 'String.Format(@"I enjoy {0}.  I am {1}.  What time is {0} served?", "lunch", "hungry")'
 cse 'ExecStmt' 'string S0 = "dinner"; string S1 = "not hungry now";'
 cse 'GetValue' 'String.Format(@"I enjoy {0}.  I am {1}.  What time is {0} served?", S0, S1)'
 cse 'Close'
```



CSE GetValueEx Method

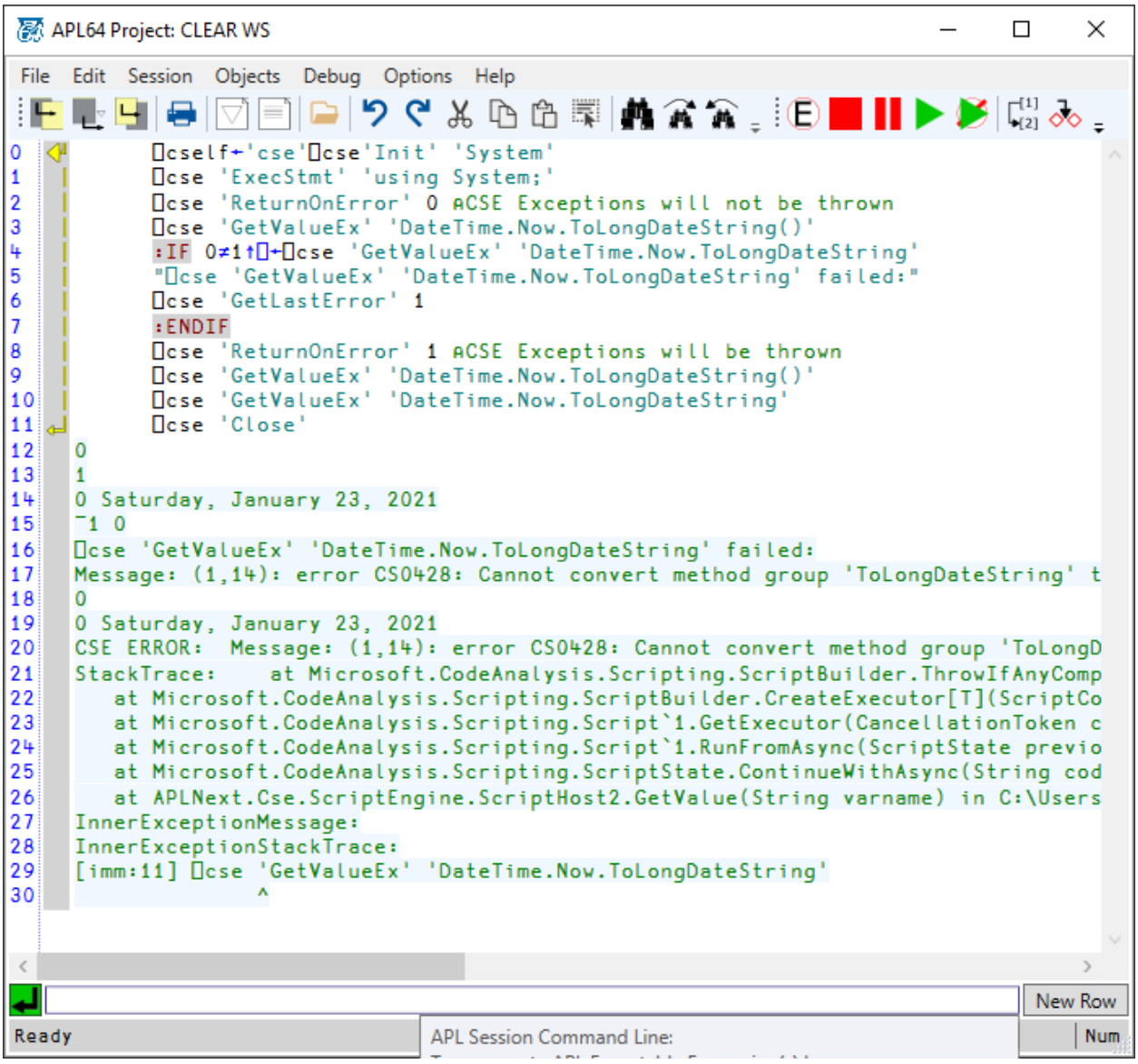
The action of the CSE GetValueEx method is the same as the CSE GetValue method, except that the result of the CSE GetValue method will contain two values when the CSE instance ‘ReturnOnError’ property is 0 (false). The CSE GetValueEx method result can therefore distinguish between an error code and the desired .Net value, if any.

Error Occurred	ReturnOnError	GetValueEx Method Result	Exception Thrown
No	0 or 1	(0, ValueOfNetObject)	No
Yes	0	(ErrorCode, 0)	No
Yes	1	n/a	Yes

Example #142

CSE instance method, GetValueEx, can enable the APL64 programmer to distinguish between a valid result and an error code.

```
□cself←'cse'□cse'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ReturnOnError' 0 ⍝CSE Exceptions will not be thrown
□cse 'GetValueEx' 'DateTime.Now.ToLongDateString()'
:IF 0≠1↑□←□cse 'GetValueEx' 'DateTime.Now.ToLongDateString'
"□cse 'GetValueEx' 'DateTime.Now.ToLongDateString' failed:"
□cse 'GetLastError' 1
:ENDIF
□cse 'ReturnOnError' 1 ⍝CSE Exceptions will be thrown
□cse 'GetValueEx' 'DateTime.Now.ToLongDateString()'
□cse 'GetValueEx' 'DateTime.Now.ToLongDateString'
□cse 'Close'
```



CSE GetVariables Method

The CSE 'GetVariables' method will return a vector of text vectors of the names of public .Net variables (fields) which exist in the specified scope. The .Net type of each variable is prefixed to the variable name.

The right arguments of the CSE 'GetVariables' method are defined as follows:

Scope of Result	Instance or Static Variables	Include Private Variables	1st Right Argument	2nd Right Argument	3rd Right Argument
Specified Class Instance	Instance	Yes	InstanceName	0	0
Specified Class Instance	Instance	Yes	InstanceName	0	1
Specified Class Instance	Static	No	ClassName	1	0
Specified Class Instance	Static	Yes	ClassName	1	1
Specified Static Class	Instance (None)	No	ClassName	1	0
Specified Static Class	Instance (None)	Yes	ClassName	1	1
Specified Static Class	Static	No	ClassName	1	0
Specified Static Class	Static	Yes	ClassName	1	1
Current CSE Instance	Instance	Yes	''	0	0
Current CSE Instance	Instance	Yes	''	0	1
Current CSE Instance	Static	Yes	''	1	0
Current CSE Instance	Static	Yes	''	1	1

The 2nd and 3rd right argument of the CSE 'GetVariables' method are optional. For example, the following statements are equivalent syntax:

- GetVariables ''
- GetVariables '' 0
- GetVariables '' 0 0

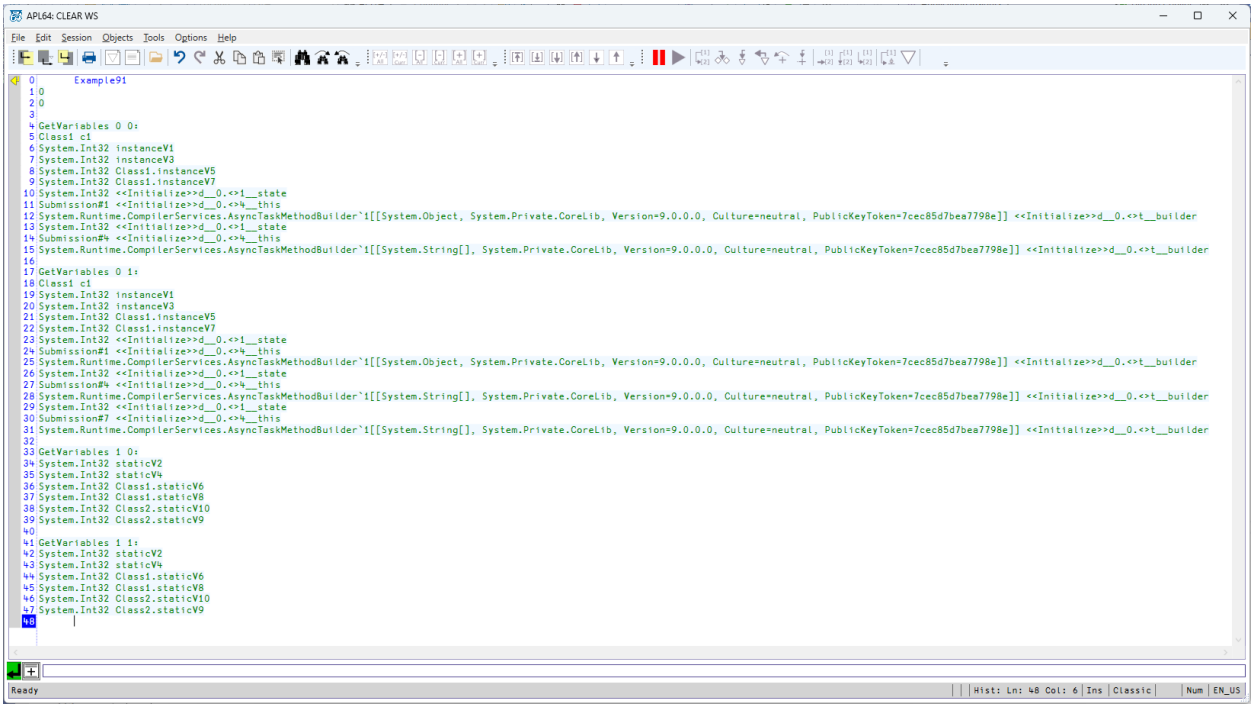
For some GetVariables action right arguments, information about internal variables of the CSE will be returned.

Example #91:
Illustration of right argument variations of the CSE 'GetVariables' method for the current CSE instance.

```
Example91
S←««
using System;
public Int32 instanceV1;
public static Int32 staticV2 = 1234;
private Int32 instanceV3;
private static Int32 staticV4 = 567;
public class Class1
```

```
{
public Int32 instanceV5;
public static Int32 staticV6 = 1234;
private Int32 instanceV7;
private static Int32 staticV8 = 567;
}
public static class Class2
{
public static Int32 staticV9 = 1234;
private static Int32 staticV10 = 567;
}
Class1 c1 = new Class1();
»»
□cself←'C' □cse 'Init' 'System'
□cse 'returnonerror' 0
□cse 'Exec' S
' '
'GetVariables 0 0:'
⊃□cse 'GetVariables' '' 0 0
' '
'GetVariables 0 1:'
⊃□cse 'GetVariables' '' 0 1
' '
'GetVariables 1 0:'
⊃□cse 'GetVariables' '' 1 0
' '
'GetVariables 1 1:'
⊃□cse 'GetVariables' '' 1 1
□cse 'Close'
```

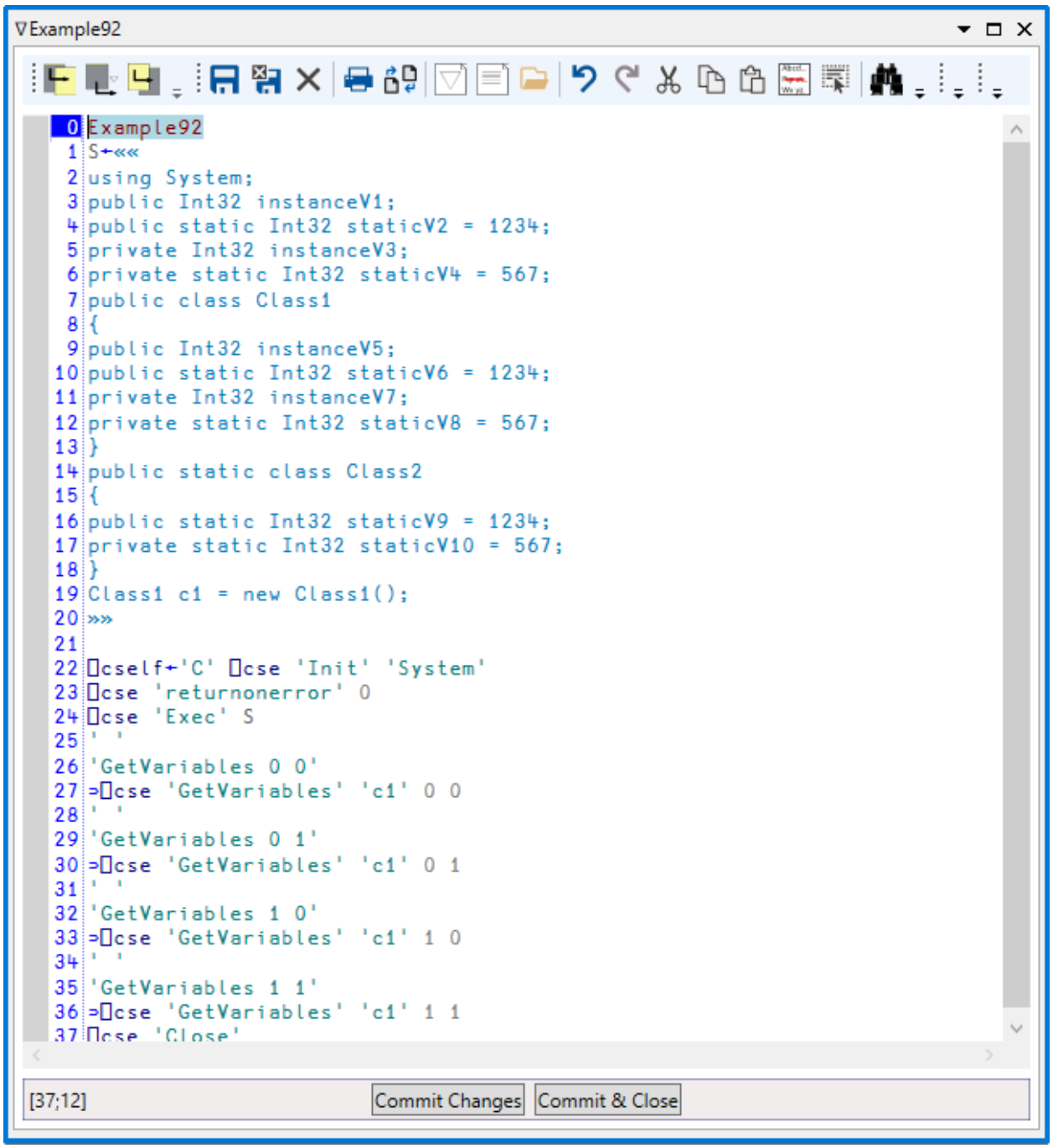

∇Example91
0 Example91
1 S←⍬⍬
2 using System;
3 public Int32 instanceV1;
4 public static Int32 staticV2 = 1234;
5 private Int32 instanceV3;
6 private static Int32 staticV4 = 567;
7 public class Class1
8 {
9 public Int32 instanceV5;
10 public static Int32 staticV6 = 1234;
11 private Int32 instanceV7;
12 private static Int32 staticV8 = 567;
13 }
14 public static class Class2
15 {
16 public static Int32 staticV9 = 1234;
17 private static Int32 staticV10 = 567;
18 }
19 Class1 c1 = new Class1();
20 ⍝⍝⍝
21 ⎕cself←'C' ⎕cse 'Init' 'System'
22 ⎕cse 'returnonerror' 0
23 ⎕cse 'Exec' S
24 ' '
25 'GetVariables 0 0:'
26 ⍝⎕cse 'GetVariables' '' 0 0
27 ' '
28 'GetVariables 0 1:'
29 ⍝⎕cse 'GetVariables' '' 0 1
30 ' '
31 'GetVariables 1 0:'
32 ⍝⎕cse 'GetVariables' '' 1 0
33 ' '
34 'GetVariables 1 1:'
35 ⍝⎕cse 'GetVariables' '' 1 1
36 ⎕cse 'Close'
37
[0;0]
Commit Changes
Commit & Close

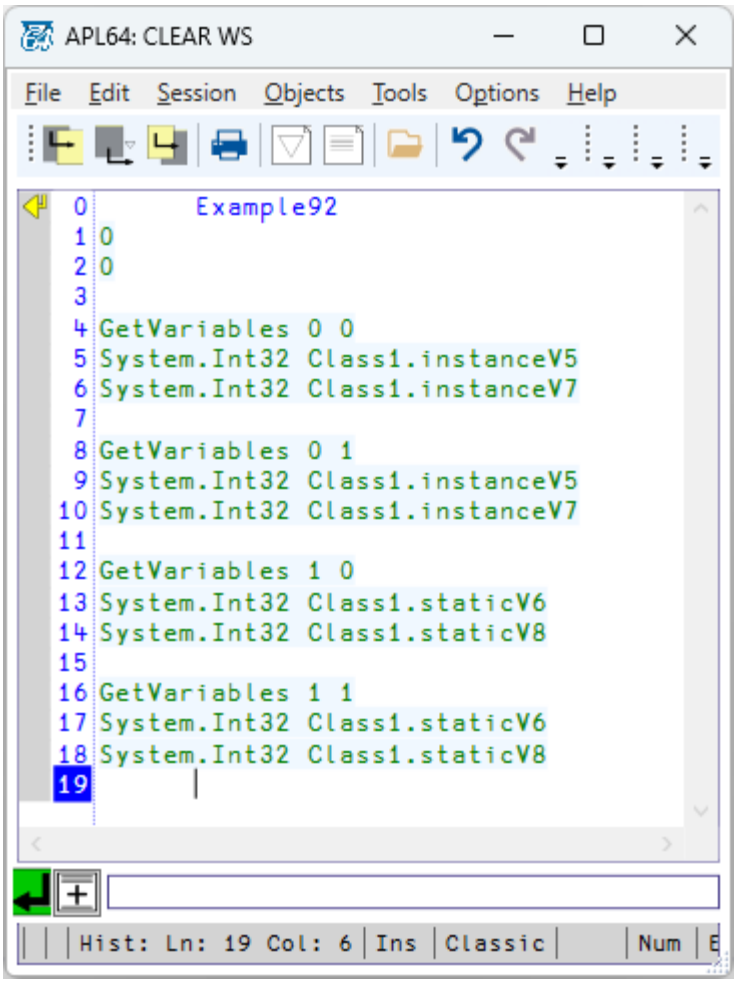


Example #92:
Illustration of right argument variations of the CSE ‘GetVariables’ method for the ‘c1’ instance of Class1.

```
Example92
S←««
using System;
public Int32 instanceV1;
public static Int32 staticV2 = 1234;
private Int32 instanceV3;
private static Int32 staticV4 = 567;
public class Class1
{
public Int32 instanceV5;
public static Int32 staticV6 = 1234;
private Int32 instanceV7;
private static Int32 staticV8 = 567;
}
public static class Class2
{
public static Int32 staticV9 = 1234;
private static Int32 staticV10 = 567;
}
```

```
Class1 c1 = new Class1();
»»

⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
⎕cse 'Exec' S
' '
'GetVariables 0 0'
⊃⎕cse 'GetVariables' 'c1' 0 0
' '
'GetVariables 0 1'
⊃⎕cse 'GetVariables' 'c1' 0 1
' '
'GetVariables 1 0'
⊃⎕cse 'GetVariables' 'c1' 1 0
' '
'GetVariables 1 1'
⊃⎕cse 'GetVariables' 'c1' 1 1
⎕cse 'Close'
```

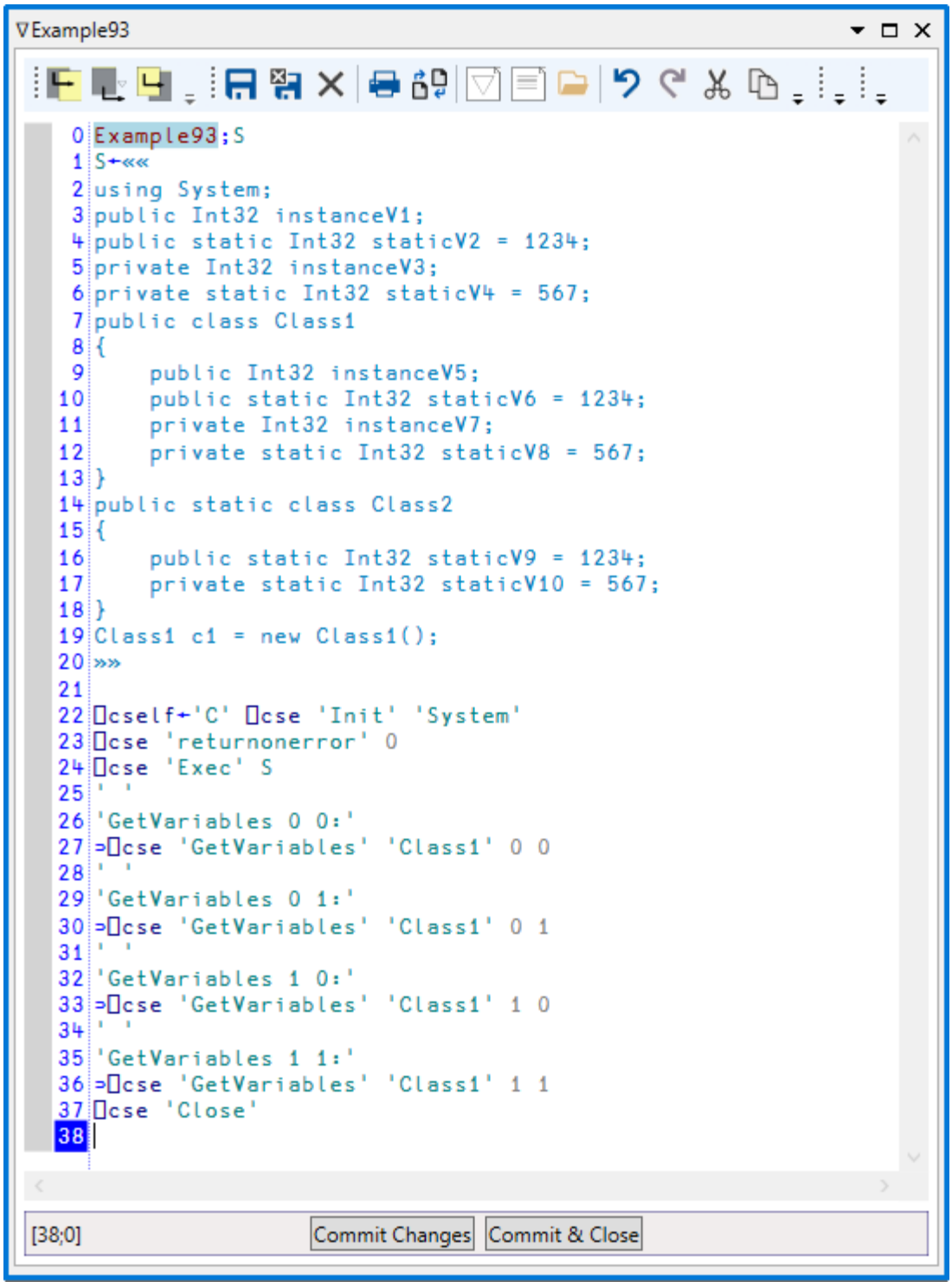

∇Example92
0 Example92
1 S←⋘
2 using System;
3 public Int32 instanceV1;
4 public static Int32 staticV2 = 1234;
5 private Int32 instanceV3;
6 private static Int32 staticV4 = 567;
7 public class Class1
8 {
9 public Int32 instanceV5;
10 public static Int32 staticV6 = 1234;
11 private Int32 instanceV7;
12 private static Int32 staticV8 = 567;
13 }
14 public static class Class2
15 {
16 public static Int32 staticV9 = 1234;
17 private static Int32 staticV10 = 567;
18 }
19 Class1 c1 = new Class1();
20 ⋙
21
22 ⎕cself←'C' ⎕cse 'Init' 'System'
23 ⎕cse 'returnonerror' 0
24 ⎕cse 'Exec' S
25 ' '
26 'GetVariables 0 0'
27 ⍎⎕cse 'GetVariables' 'c1' 0 0
28 ' '
29 'GetVariables 0 1'
30 ⍎⎕cse 'GetVariables' 'c1' 0 1
31 ' '
32 'GetVariables 1 0'
33 ⍎⎕cse 'GetVariables' 'c1' 1 0
34 ' '
35 'GetVariables 1 1'
36 ⍎⎕cse 'GetVariables' 'c1' 1 1
37 ⎕cse 'Close'
[37;12]
Commit Changes
Commit & Close

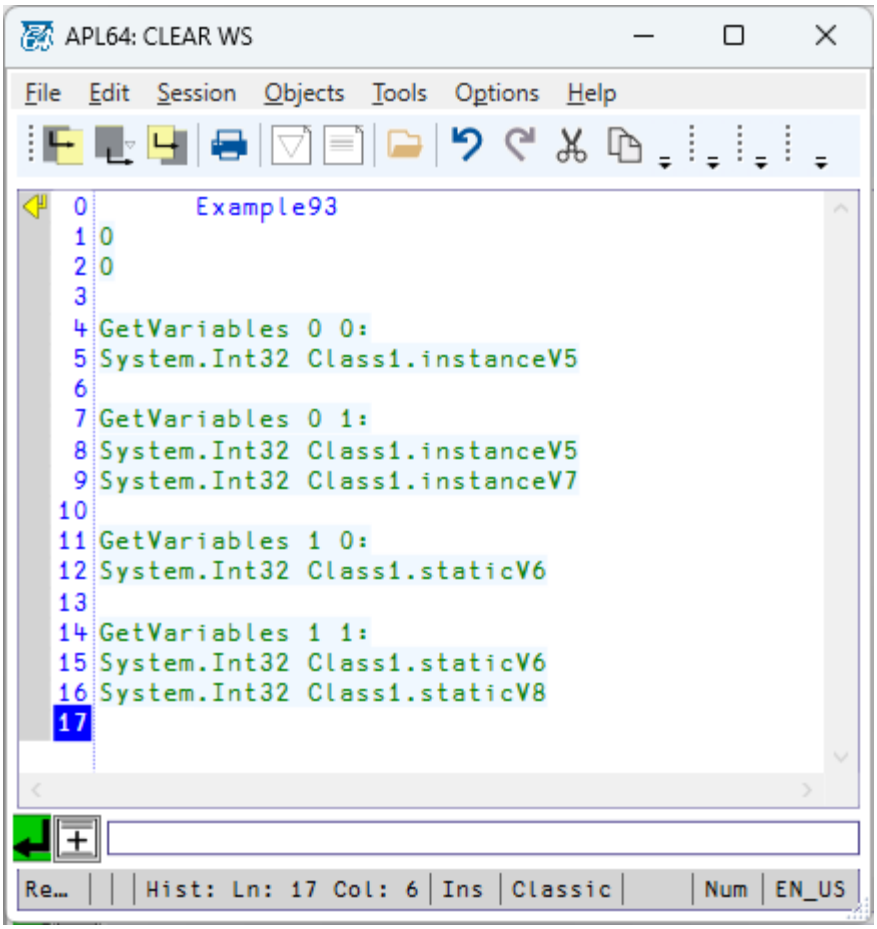


Example #93:
Illustration of right argument variations of the CSE ‘GetVariables’ method for the ‘Class1’ class name.

```
Example93;S
S←««
using System;
public Int32 instanceV1;
public static Int32 staticV2 = 1234;
private Int32 instanceV3;
private static Int32 staticV4 = 567;
public class Class1
{
   public Int32 instanceV5;
   public static Int32 staticV6 = 1234;
   private Int32 instanceV7;
   private static Int32 staticV8 = 567;
}
```

```
public static class Class2
{
   public static Int32 staticV9 = 1234;
   private static Int32 staticV10 = 567;
}
Class1 c1 = new Class1();
»»

⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
⎕cse 'Exec' S
' '
'GetVariables 0 0:'
⊃⎕cse 'GetVariables' 'Class1' 0 0
' '
'GetVariables 0 1:'
⊃⎕cse 'GetVariables' 'Class1' 0 1
' '
'GetVariables 1 0:'
⊃⎕cse 'GetVariables' 'Class1' 1 0
' '
'GetVariables 1 1:'
⊃⎕cse 'GetVariables' 'Class1' 1 1
⎕cse 'Close'
```

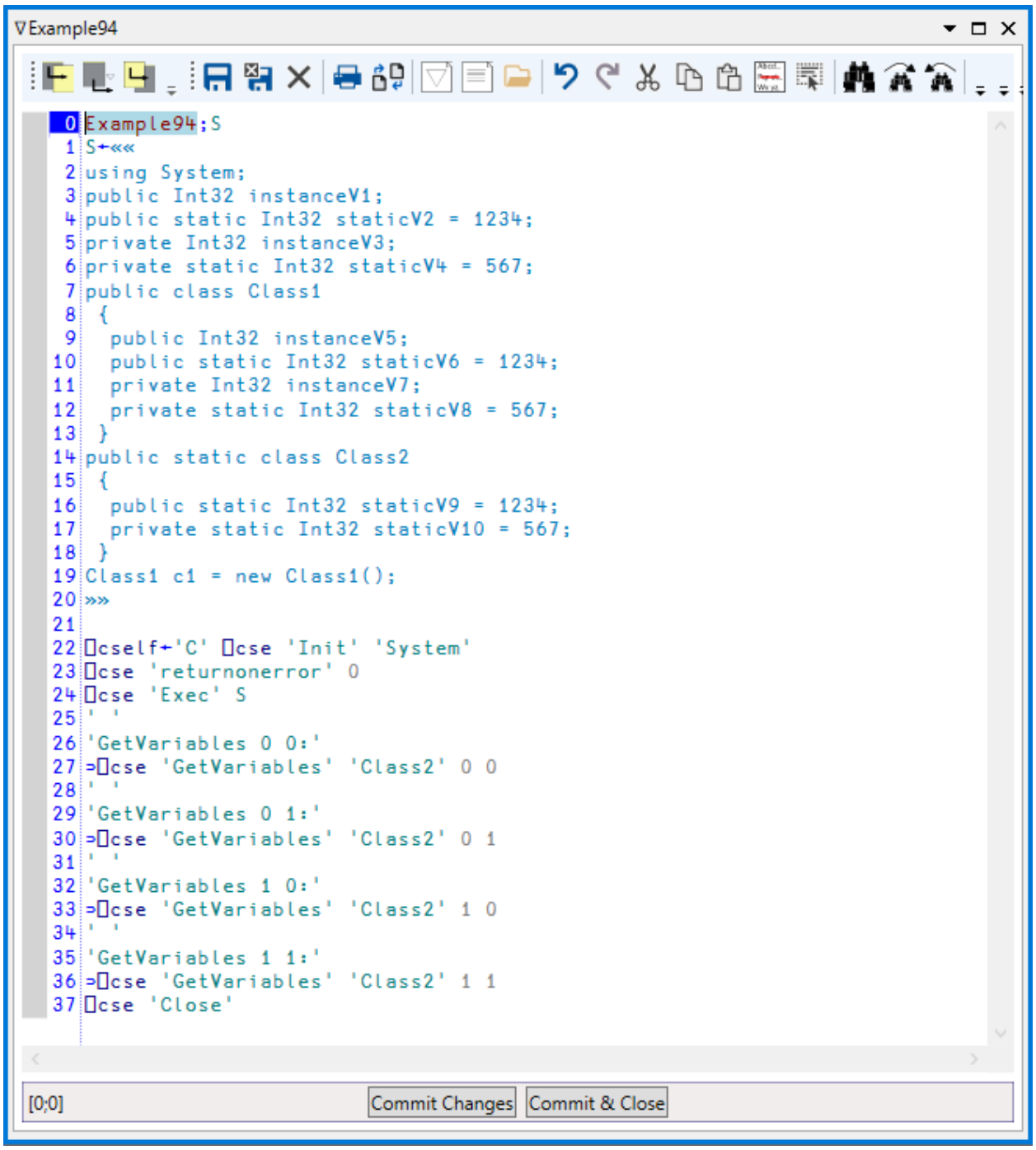

∇Example93
0 Example93;S
1 S←⍬⍬
2 using System;
3 public Int32 instanceV1;
4 public static Int32 staticV2 = 1234;
5 private Int32 instanceV3;
6 private static Int32 staticV4 = 567;
7 public class Class1
8 {
9 public Int32 instanceV5;
10 public static Int32 staticV6 = 1234;
11 private Int32 instanceV7;
12 private static Int32 staticV8 = 567;
13 }
14 public static class Class2
15 {
16 public static Int32 staticV9 = 1234;
17 private static Int32 staticV10 = 567;
18 }
19 Class1 c1 = new Class1();
20 ⍝⍝
21
22 ⎕cself←'C' ⎕cse 'Init' 'System'
23 ⎕cse 'returnonerror' 0
24 ⎕cse 'Exec' S
25 ' '
26 'GetVariables 0 0:'
27 ⍕⎕cse 'GetVariables' 'Class1' 0 0
28 ' '
29 'GetVariables 0 1:'
30 ⍕⎕cse 'GetVariables' 'Class1' 0 1
31 ' '
32 'GetVariables 1 0:'
33 ⍕⎕cse 'GetVariables' 'Class1' 1 0
34 ' '
35 'GetVariables 1 1:'
36 ⍕⎕cse 'GetVariables' 'Class1' 1 1
37 ⎕cse 'Close'
38
[38;0]
Commit Changes
Commit & Close

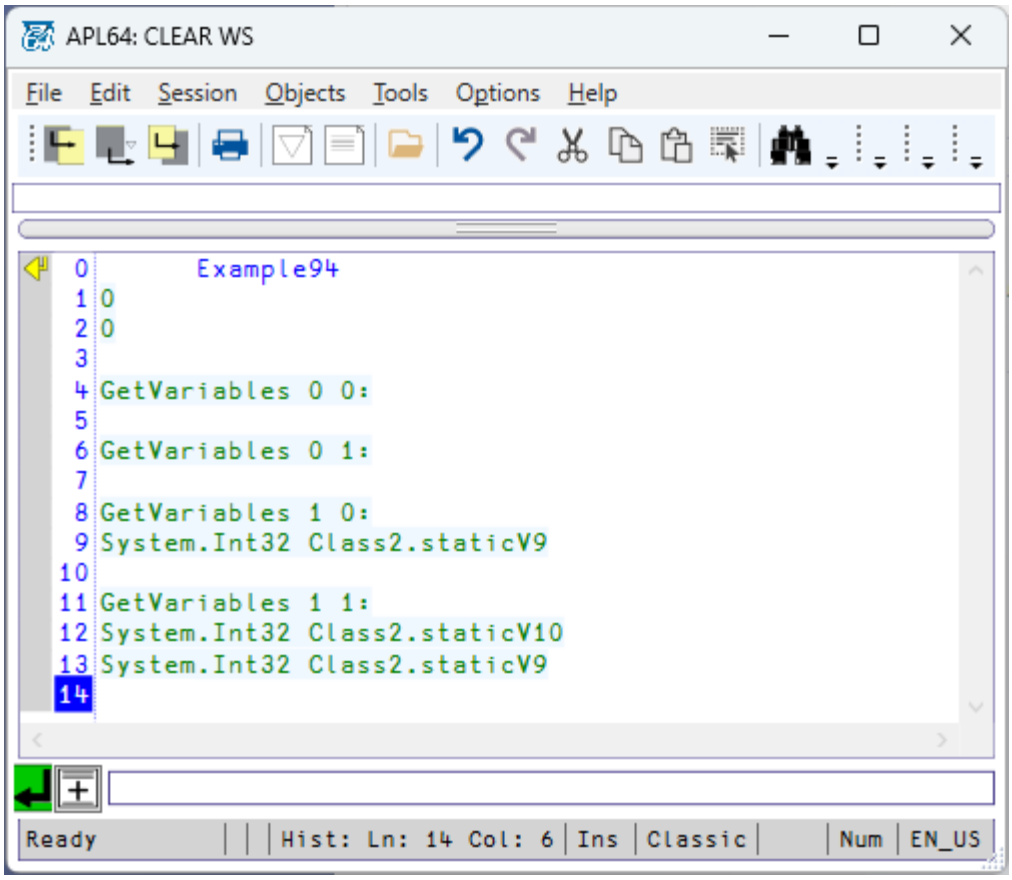


Example #94:
Illustration of right argument variations of the CSE ‘GetVariables’ method for the ‘Class2’ class name.

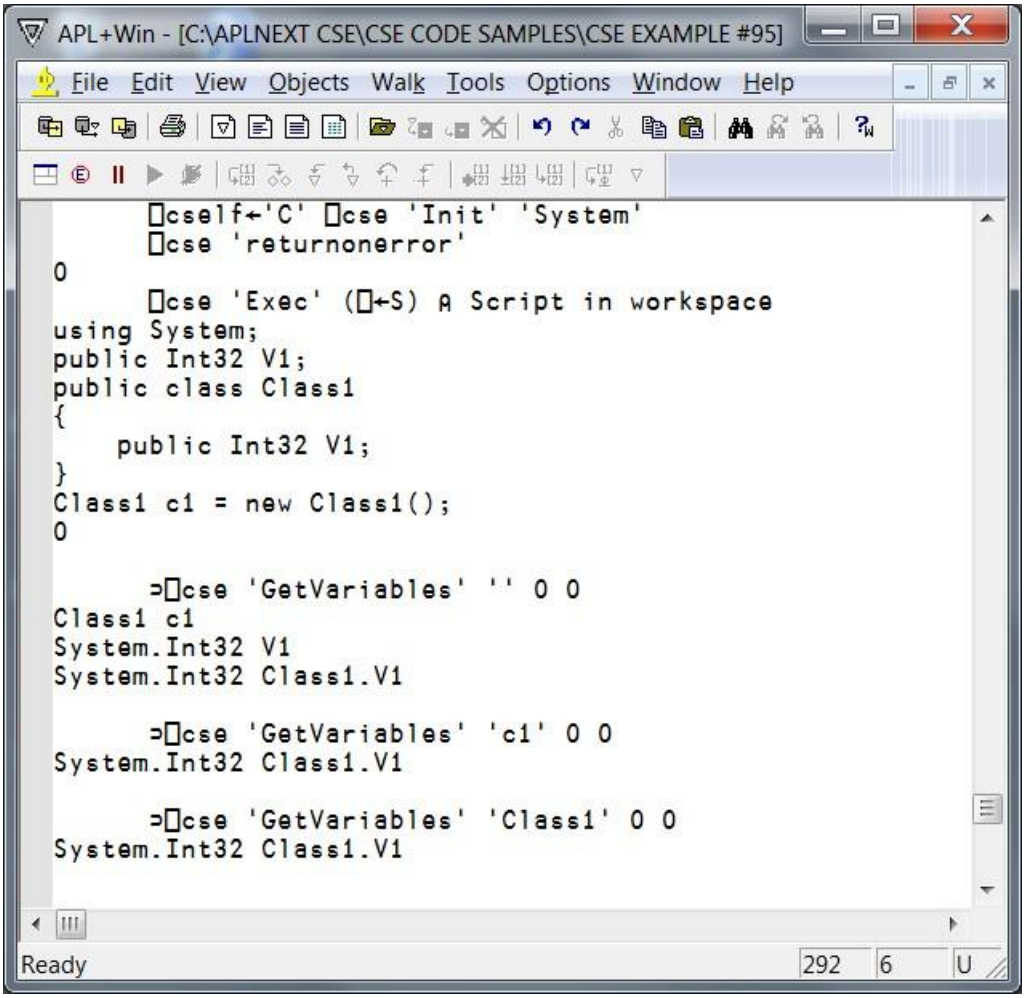
```
Example94;S
S←««
using System;
public Int32 instanceV1;
public static Int32 staticV2 = 1234;
private Int32 instanceV3;
private static Int32 staticV4 = 567;
public class Class1
 {
  public Int32 instanceV5;
  public static Int32 staticV6 = 1234;
  private Int32 instanceV7;
  private static Int32 staticV8 = 567;
 }
public static class Class2
 {
```

```
  public static Int32 staticV9 = 1234;
  private static Int32 staticV10 = 567;
 }
Class1 c1 = new Class1();
»»

⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
⎕cse 'Exec' S
' '
'GetVariables 0 0:'
⊃⎕cse 'GetVariables' 'Class2' 0 0
' '
'GetVariables 0 1:'
⊃⎕cse 'GetVariables' 'Class2' 0 1
' '
'GetVariables 1 0:'
⊃⎕cse 'GetVariables' 'Class2' 1 0
' '
'GetVariables 1 1:'
⊃⎕cse 'GetVariables' 'Class2' 1 1
⎕cse 'Close'
```


VExample94
0 Example94;S
1 S←⋄⋄
2 using System;
3 public Int32 instanceV1;
4 public static Int32 staticV2 = 1234;
5 private Int32 instanceV3;
6 private static Int32 staticV4 = 567;
7 public class Class1
8 {
9 public Int32 instanceV5;
10 public static Int32 staticV6 = 1234;
11 private Int32 instanceV7;
12 private static Int32 staticV8 = 567;
13 }
14 public static class Class2
15 {
16 public static Int32 staticV9 = 1234;
17 private static Int32 staticV10 = 567;
18 }
19 Class1 c1 = new Class1();
20 ⋄⋄
21
22 □cself←'C' □cse 'Init' 'System'
23 □cse 'returnonerror' 0
24 □cse 'Exec' S
25 ' '
26 'GetVariables 0 0:'
27 ⎕←□cse 'GetVariables' 'Class2' 0 0
28 ' '
29 'GetVariables 0 1:'
30 ⎕←□cse 'GetVariables' 'Class2' 0 1
31 ' '
32 'GetVariables 1 0:'
33 ⎕←□cse 'GetVariables' 'Class2' 1 0
34 ' '
35 'GetVariables 1 1:'
36 ⎕←□cse 'GetVariables' 'Class2' 1 1
37 □cse 'Close'
[0;0]
Commit Changes
Commit & Close

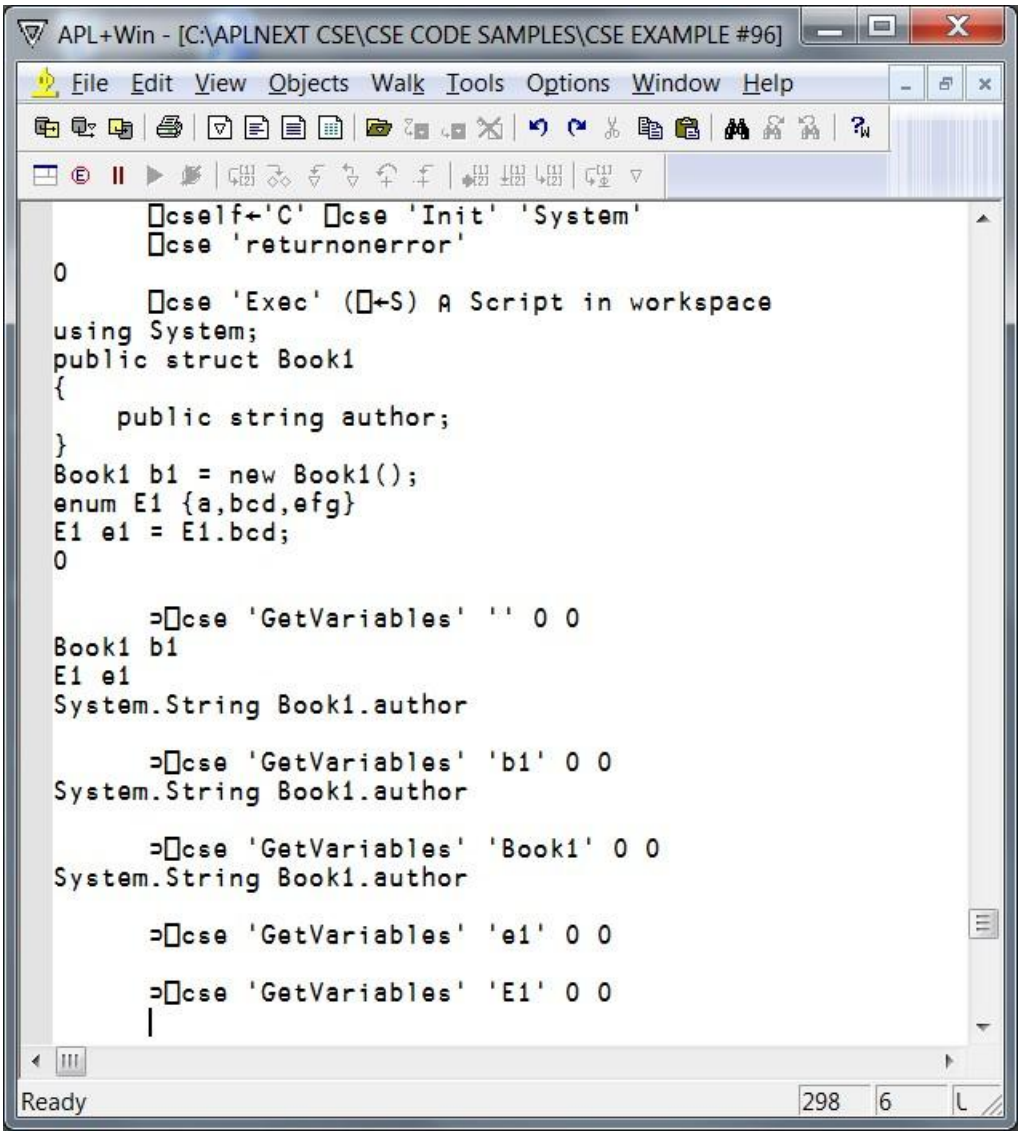


Example #95:
In this example two variables with the same name exist, one in the scope of the current CSE instance and one in the scope of a class instance, ‘c1’ of Class1:



Example #96

In this example a C# struct and a C# enumeration are illustrated:



CSE Init Method

The CSE 'Init' method creates an instance of an APLNext CSE object in the memory space external to the APL64 workspace. Multiple instances of CSE objects may be created up to the memory limits of the workstation containing the CSE software.

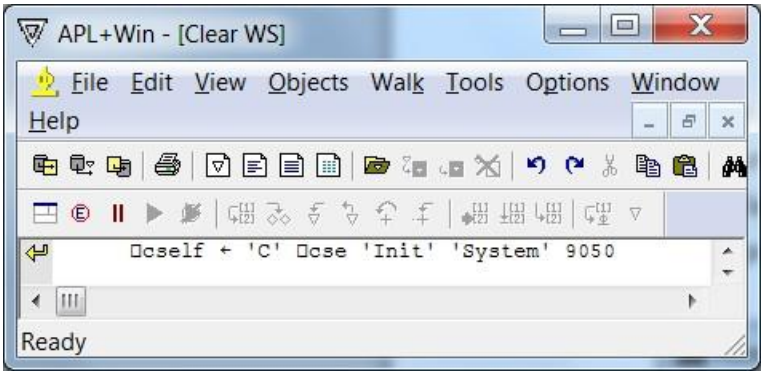
The left argument of the CSE 'Init' method is an APL64 text scalar or text vector providing the APL64 name of the CSE instance being initialized or □cself with a previously set value. Each CSE instance created during an APL64 session must have a unique CSE instance name.

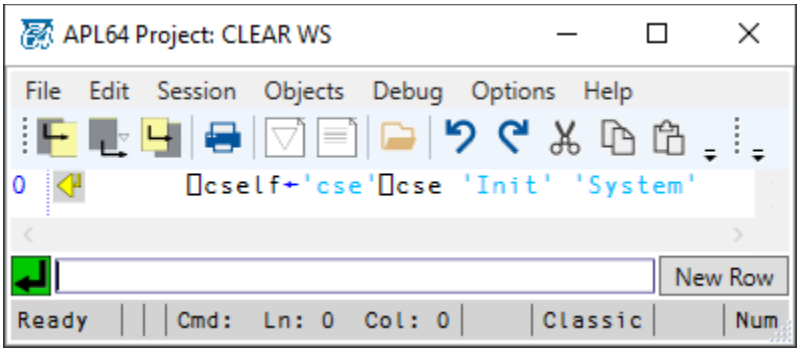
The CSE 'Init' method returns a text vector containing the APL64 name associated with that instance of the CSE object.

The required right argument of the 'Init' method is an APL64 string or vector of text vectors which provides the names of one or more .Net assemblies to be loaded into the new instance of a CSE object.

- Generally, the Microsoft .Net 'System' assembly should always be included in the .Net assemblies loaded into a CSE instance.
- Some .Net assemblies contain more than one .Net namespace and some .Net namespaces, e.g. 'System', are contained in more than one .Net assembly. More information is available here and here. When .Net is installed on a workstation, the '…\windows\assembly\' folder will contain all assemblies in the Global Assembly Cache (GAC), but the structure of this folder is not amenable to direct examination.

In the APL+Win an optional port number could be provided in the right argument of the APL+Win 'Init' method. The port number argument is not needed and should not be provided in the right argument of the APL64 CSE 'Init' method.



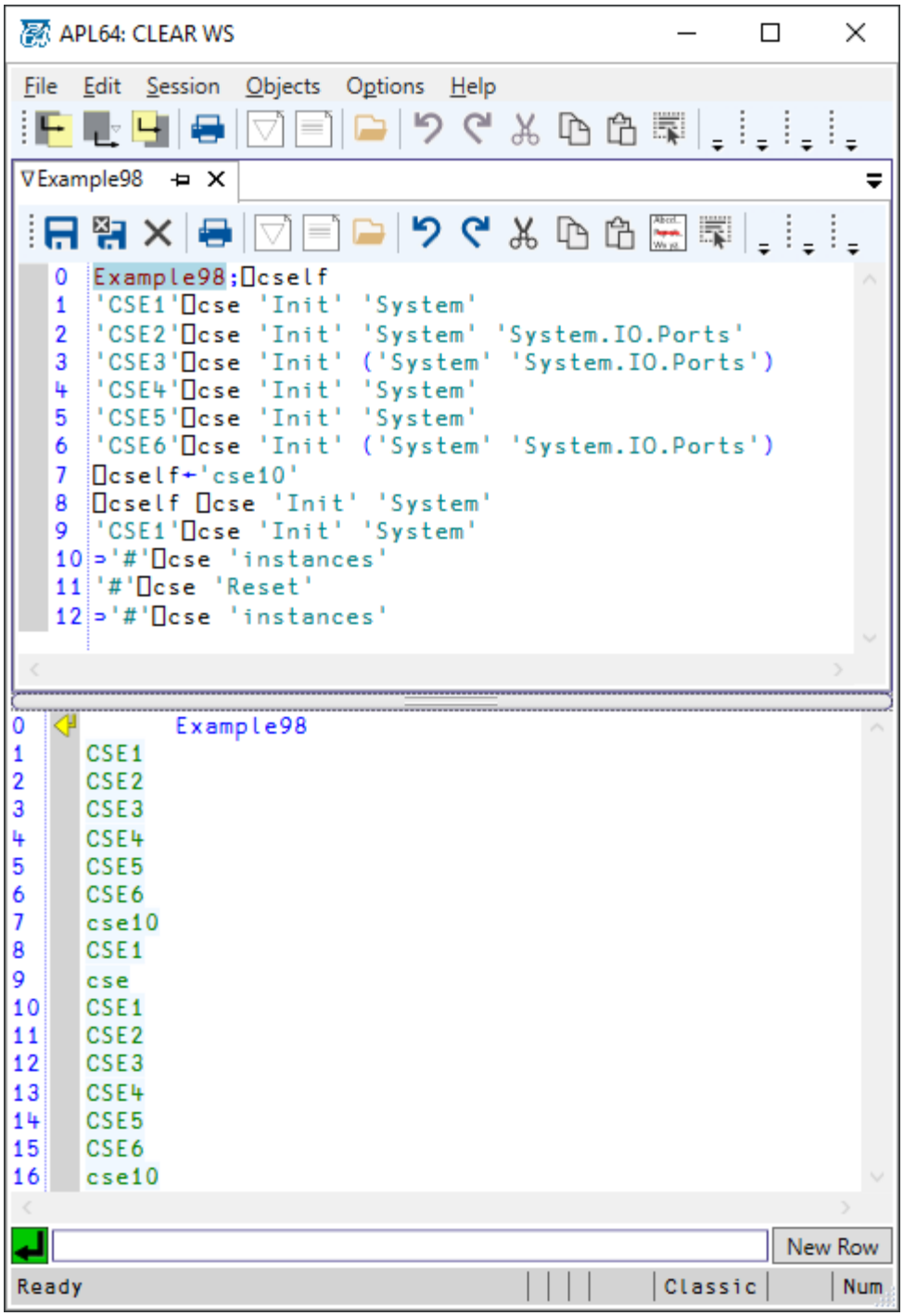


If the CSE 'Init' method is called using the same name (left argument) as that for a pre-existing CSE instance, that prior instance will be closed and a new CSE instance with the selected name will be initialized.

Example #98

```
Example98;□cself
'CSE1'□cse 'Init' 'System'
'CSE2'□cse 'Init' 'System' 'System.IO.Ports'
'CSE3'□cse 'Init' ('System' 'System.IO.Ports')
```

```
'CSE4'⎕cse 'Init' 'System'
'CSE5'⎕cse 'Init' 'System'
'CSE6'⎕cse 'Init' ('System' 'System.IO.Ports')
⎕cself←'cse10'
⎕cself ⎕cse 'Init' 'System'
'CSE1'⎕cse 'Init' 'System'
⊃'#'⎕cse 'instances'
'#'⎕cse 'Reset'
⊃'#'⎕cse 'instances'
```

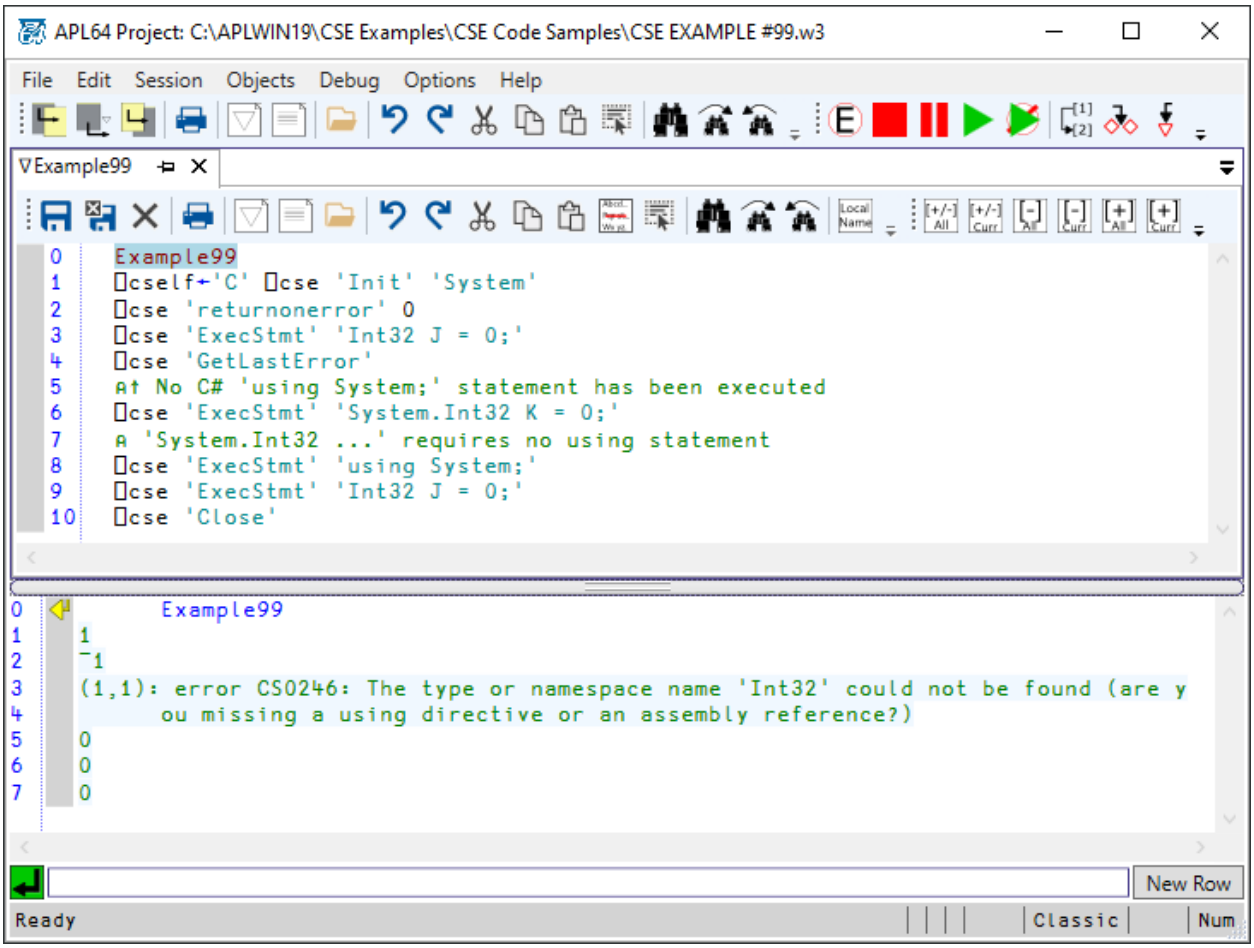


Example #99:

Loading a .Net assembly does not automatically create a C# ‘using’ statement for the .Net namespaces associated with that .Net assembly. Use the CSE ‘Exec’, ‘ExecStmt’ or ‘ExecFile’ method to execute a C# ‘using’ statement. A C# using statement is convenient because then it is not necessary to enter the fully qualified name of a .Net object in the C# source code.

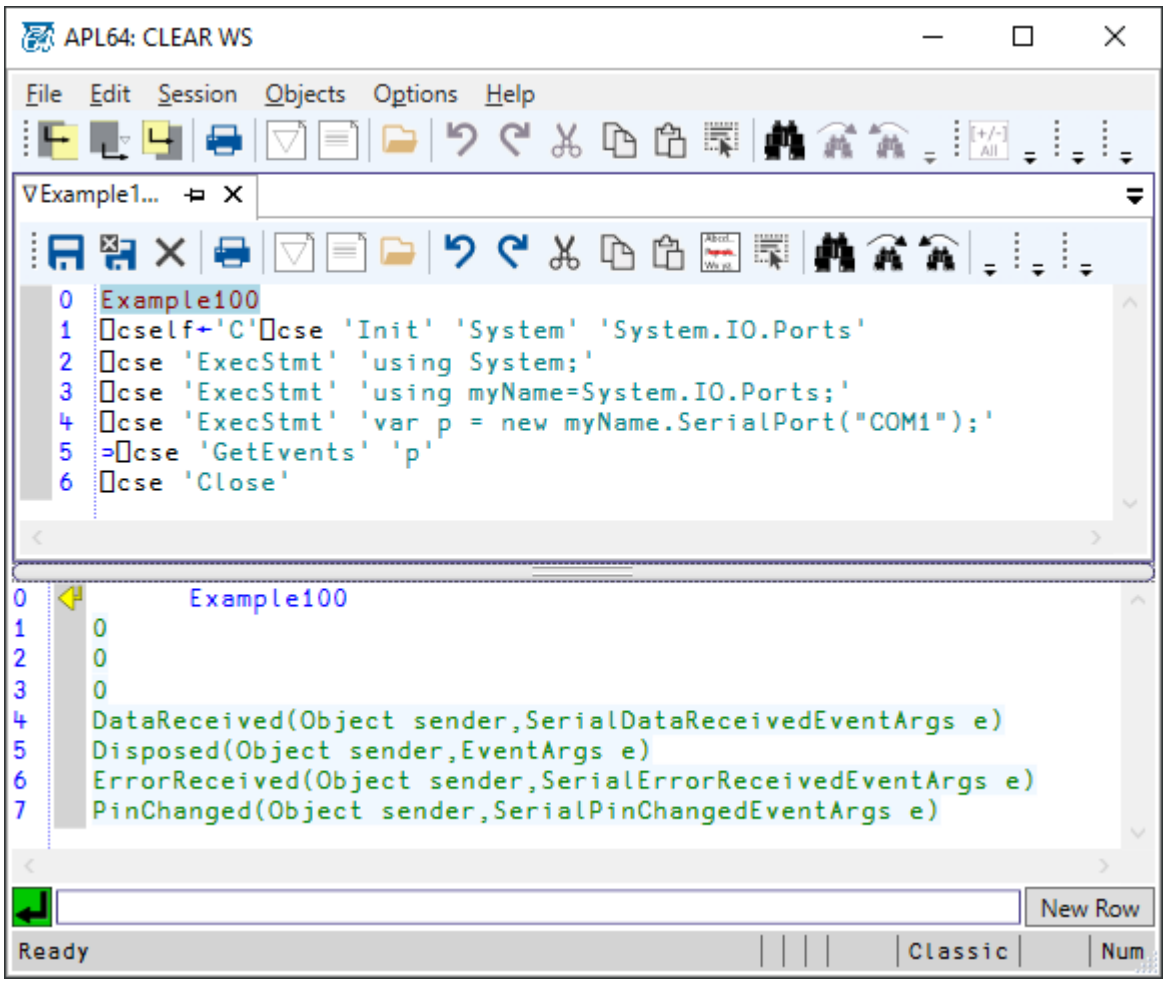
The ‘Int32’ .Net type is defined within the ‘System’ namespace as ‘System.Int32’. Without executing the ‘using System;’ statement, the C# debugger cannot parse ‘Int32 J = 0;’, but it can parse ‘System.Int32 K = 10;’ because the latter statement uses the fully qualified type name. After executing ‘using System;’ the statement ‘Int32 J = 0;’ can be parsed.

```
Example99
□cself←'C' □cse 'Init' 'System'
□cse 'returnonerror' 0
□cse 'ExecStmt' 'Int32 J = 0;'
□cse 'GetLastError'
⍝↑ No C# 'using System;' statement has been executed
□cse 'ExecStmt' 'System.Int32 K = 0;'
⍝ 'System.Int32 ...' requires no using statement
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'Int32 J = 0;'
□cse 'Close'
```



Example #100:
It is also possible to execute a using statement to define a ‘custom surrogate’ for a namespace. In this example the text ‘myName’ is the surrogate for the ‘System.IO.Ports’ namespace.

```
Example100
 cself←'C' cse 'Init' 'System' 'System.IO.Ports'
 cse 'ExecStmt' 'using System;'
 cse 'ExecStmt' 'using myName=System.IO.Ports;'
 cse 'ExecStmt' 'var p = new myName.SerialPort("COM1");'
⊃ cse 'GetEvents' 'p'
 cse 'Close'
```



CSE InitSTA Method

The CSE ‘InitSTA’ method has been deprecated and has the same argument structure and action as the CSE Init method.

When the CSE is used in an APL64 instance, the application domain is set to 'Single Threaded Apartment State' (STA).

CSE LoadAssembly Method

A .Net assembly is a collection of .Net types and resources that forms a logical unit of functionality. A .Net assembly is contained in a file with the file extension .dll or .exe. Loading an assembly into an instance of the CSE object means that a memory-based copy of that assembly's content is now in the memory space of that CSE object and can be used in that instance of the CSE. Only .Net assemblies can be loaded using the CSE 'LoadAssembly' method.

The required right argument of the 'LoadAssembly' method is a string which specifies the desired assembly by its path. When the path provided is not fully qualified, Microsoft .Net searches for the assembly locally, in the folder of the calling environment or in the global assembly cache (GAC).

The 'LoadAssembly' method will return the .Net strong name properties of the assembly including Name, Version, Culture and Public Key token or an APL64 error will be thrown if not successful and □dm will contain the .Net error message. Note that the .Net error message if an assembly cannot be loaded may be quite verbose. If the Public Key token of the assembly is empty, that indicates that the assembly does not have a strong name.

More than one assembly can be loaded into a C# Script Engine instance using separate 'LoadAssembly' CSE method calls.

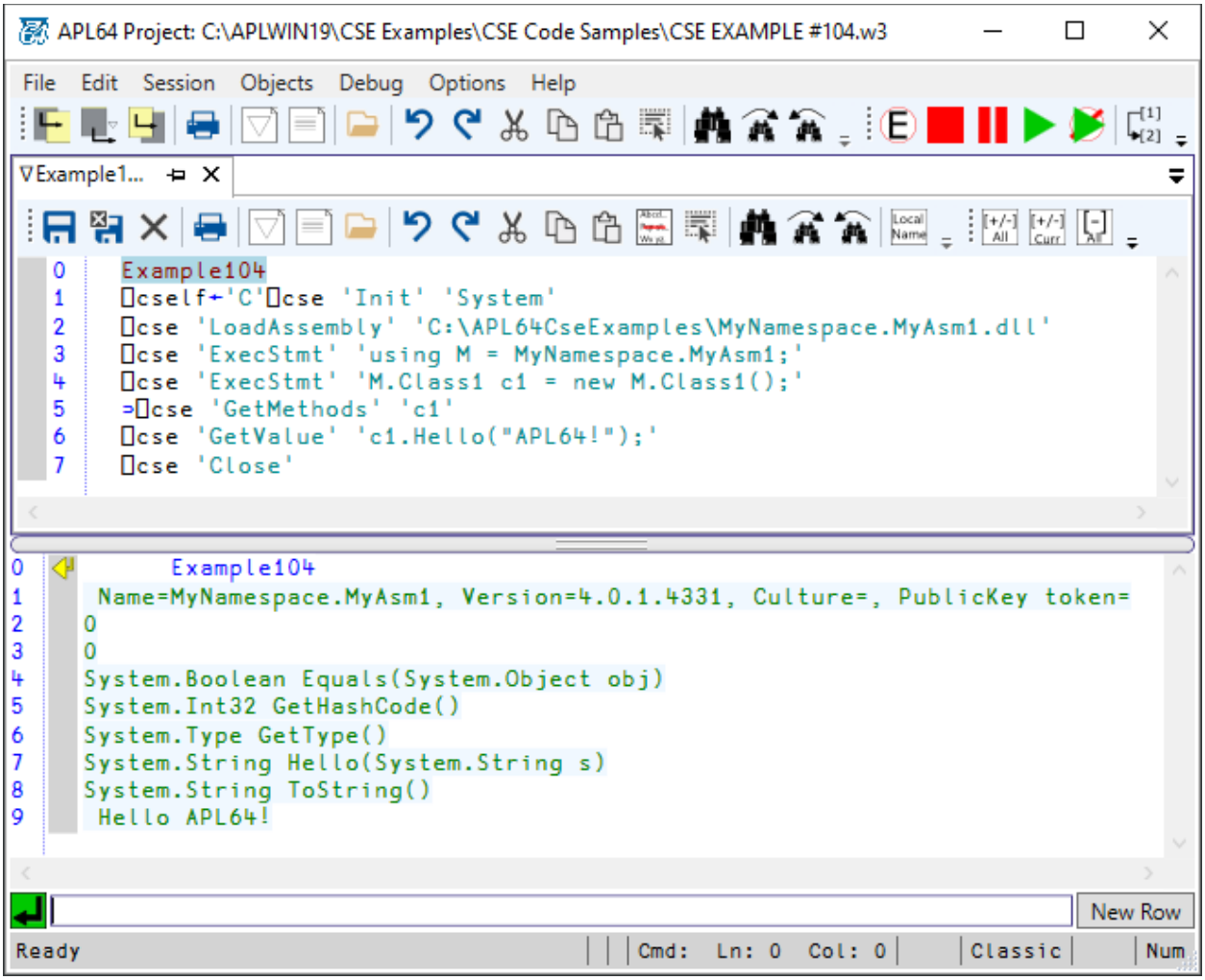
Just as with the CSE 'Init' method, the 'LoadAssembly' method does not add a 'using' statement for a namespace associated with the loaded assembly.

Pre-compiled .Net assemblies loaded by the CSE must be compiled for the applicable Microsoft Windows operating system version [64- bit] or they may be compiled for 'Any CPU'.

Example #104:

This example uses a custom C# assembly provided in the 'CSE Code Samples' folder for the CSE. The source code for this C# assembly is also available in the same folder. In this example the target assembly is loaded in to the CSE instance. The assembly's Name and Version property values are returned. The empty PublicKey Token property value indicates that this .Net assembly does not have a strong name.

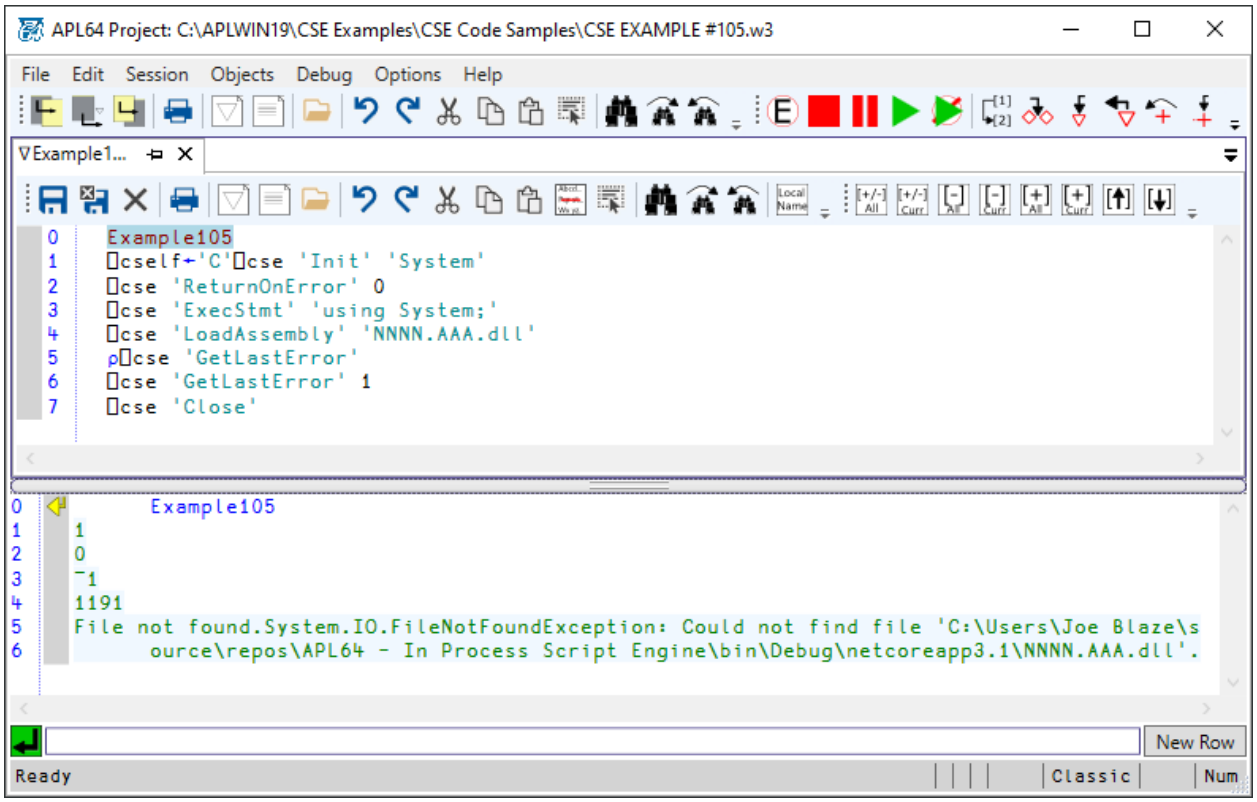
```
Example104
□cself←'C'□cse 'Init' 'System'
□cse 'LoadAssembly' 'C:\APL64CseExamples\MyNamespace.MyAsm1.dll'
□cse 'ExecStmt' 'using M = MyNamespace.MyAsm1;'
□cse 'ExecStmt' 'M.Class1 c1 = new M.Class1();'
⊃□cse 'GetMethods' 'c1'
□cse 'GetValue' 'c1.Hello("APL64!");'
□cse 'Close'
```


APL64 Project: C:\APLWIN19\CSE Examples\CSE Code Samples\CSE EXAMPLE #104.w3
File Edit Session Objects Debug Options Help
Example1...
0 Example104
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'LoadAssembly' 'C:\APL64CseExamples\MyNamespace.MyAsm1.dll'
3 ⎕cse 'ExecStmt' 'using M = MyNamespace.MyAsm1;'
4 ⎕cse 'ExecStmt' 'M.Class1 c1 = new M.Class1();'
5 ⎕cse 'GetMethods' 'c1'
6 ⎕cse 'GetValue' 'c1.Hello("APL64!");'
7 ⎕cse 'Close'
0 Example104
1 Name=MyNamespace.MyAsm1, Version=4.0.1.4331, Culture=, PublicKey token=
2 0
3 0
4 System.Boolean Equals(System.Object obj)
5 System.Int32 GetHashCode()
6 System.Type GetType()
7 System.String Hello(System.String s)
8 System.String ToString()
9 Hello APL64!
New Row
Ready
Cmd: Ln: 0 Col: 0
Classic
Num

Example #105

In this example an attempt is made to load a .Net assembly which does not exist, so that the .Net error message is returned, which quite verbose.

```
Example105
 cself←'C' cse 'Init' 'System'
 cse 'ReturnOnError' 0
 cse 'ExecStmt' 'using System;'
 cse 'LoadAssembly' 'NNNN.AAA.dll'
ρ cse 'GetLastError'
 cse 'GetLastError' 1
 cse 'Close'
```



CSE LoadAssemblyByName Method

The CSE 'LoadAssemblyByName' method will make the contents of a .Net assembly accessible in the instance of the CSE object. The required right argument of the 'LoadAssemblyByName' method is a string which specifies the desired assembly by its .Net name. This is useful when the desired assembly is located in the Global Assembly Cache (GAC) on the target machine. Only .Net assemblies can be loaded using the CSE 'LoadAssemblyByName' method.

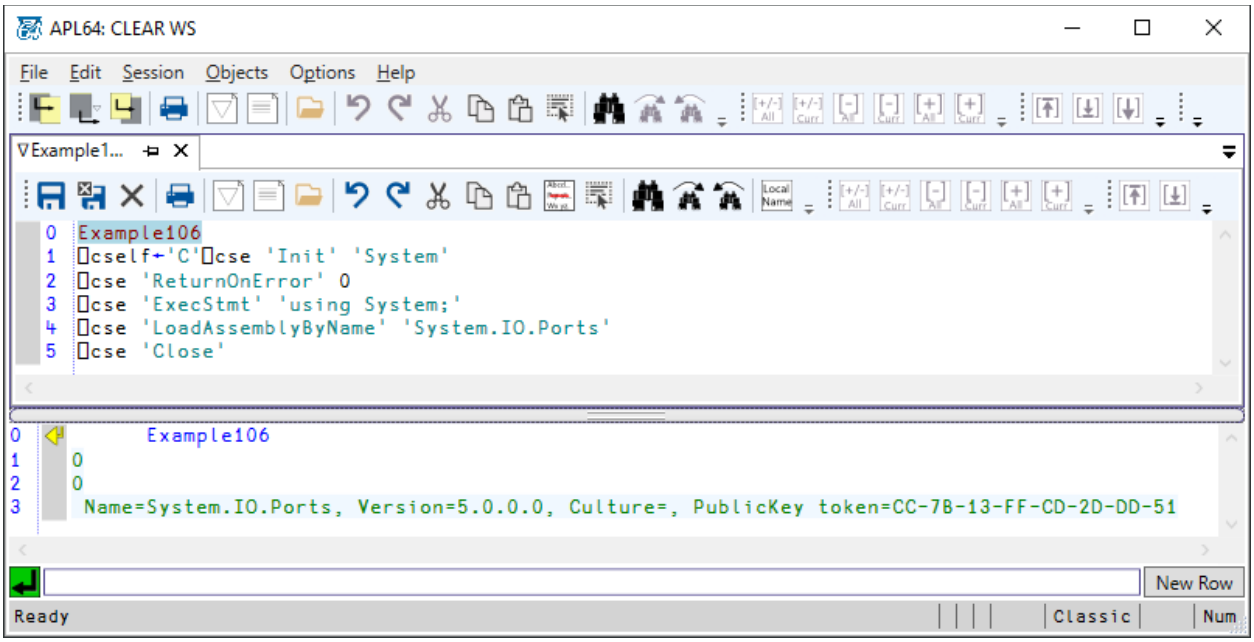
The ‘LoadAssemblyByName’ method will return the .Net strong name properties of the assembly including Name, Version, Culture and Public Key token or an APL64 error will be thrown if not successful and □dm will contain the .Net error message.

More than one assembly can be loaded into a C# Script Engine instance using separate ‘LoadAssemblyByName’ CSE method calls.

Refer to the documentation of the CSE ‘LoadAssembly’ for more information about the use of loaded assemblies.

Example #106:
Successful load of assembly by name.

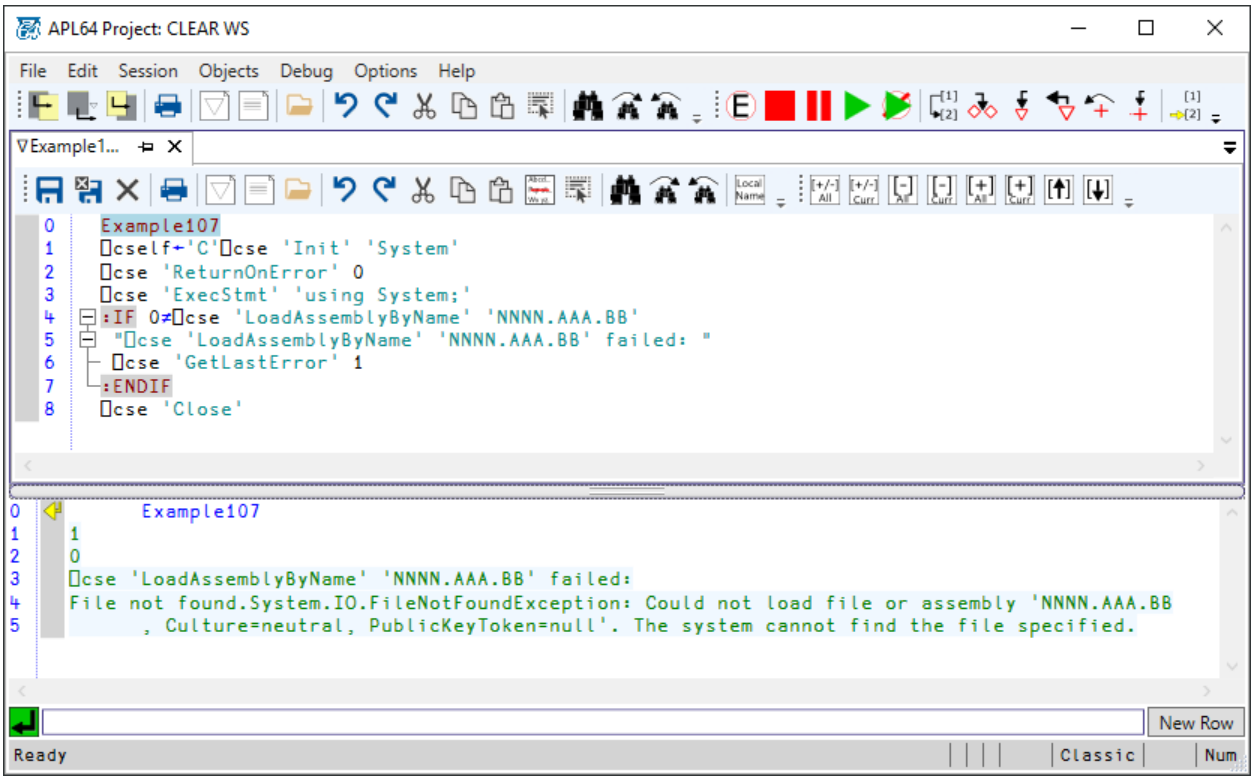
```
Example106
□cself←'C'□cse 'Init' 'System'
□cse 'ReturnOnError' 0
□cse 'ExecStmt' 'using System;'
□cse 'LoadAssemblyByName' 'System.IO.Ports'
□cse 'Close'
```



Example #107:
An attempt is made to load an assembly which does not exist using the CSE ‘LoadAssemblyByName’ method.

```
Example107
□cself←'C'□cse 'Init' 'System'
```

```
 cse 'ReturnOnError' 0
 cse 'ExecStmt' 'using System;'
:IF 0≠ cse 'LoadAssemblyByName' 'NNNN.AAA.BB'
 " cse 'LoadAssemblyByName' 'NNNN.AAA.BB' failed: "
  cse 'GetLastError' 1
:ENDIF
 cse 'Close'
```



CSE RemoveCustomEventHandler Method

This method will remove the subscription of an APL64 event handler function to a .Net custom event.

The result of the CSE 'RemoveCustomEventHandler' method is affected by the value of the CSE 'returnonerror' property.

The right argument of this method may contain up to three elements each of which is an APL64 text vector:

- Element #1 is required to contain the name of the C# object which exposed the event to which the APL64 event handler function is subscribed, e.g. 'myClass'
- Element #2 is optional and if present contains the name of the C# event exposed by the specified C# object, e.g. 'OnMyCustomEvent'

- Element #3 is optional and if present contains the name of the APL64 event handler function subscribed to the C# event, e.g. 'EHfn'

The action of the CSE 'RemoveCustomEventHandler' method depends on the number of arguments provided to the method:

- If only argument element #1 is present, the action of the CSE 'RemoveCustomEventHandler' method is to remove all the APL64 custom event handler subscriptions for the C# object specified in argument element #1
- If only argument elements #1 and #2 are present, the action of the CSE 'RemoveCustomEventHandler' method is to remove all the APL64 custom event handler subscriptions for the C# object specified in argument element #1 which are for the C# custom event specified in argument element #2
- If all three argument elements are present, the action of the CSE 'RemoveCustomEventHandler' method is to remove the APL64 custom event handler subscription for the specified C# object in argument element #1 which are for the C# custom event specified in argument element #2 which are for the APL64 custom event handler function specified in argument element #3.

CSE RemoveEventHandler Method

This method will remove the subscription of an APL64 event handler function to a .Net routed event.

The result of the CSE 'RemoveEventHandler' method is affected by the value of the CSE 'returnonerror' property.

The right argument of this method may contain up to three elements each of which is an APL64 text vector:

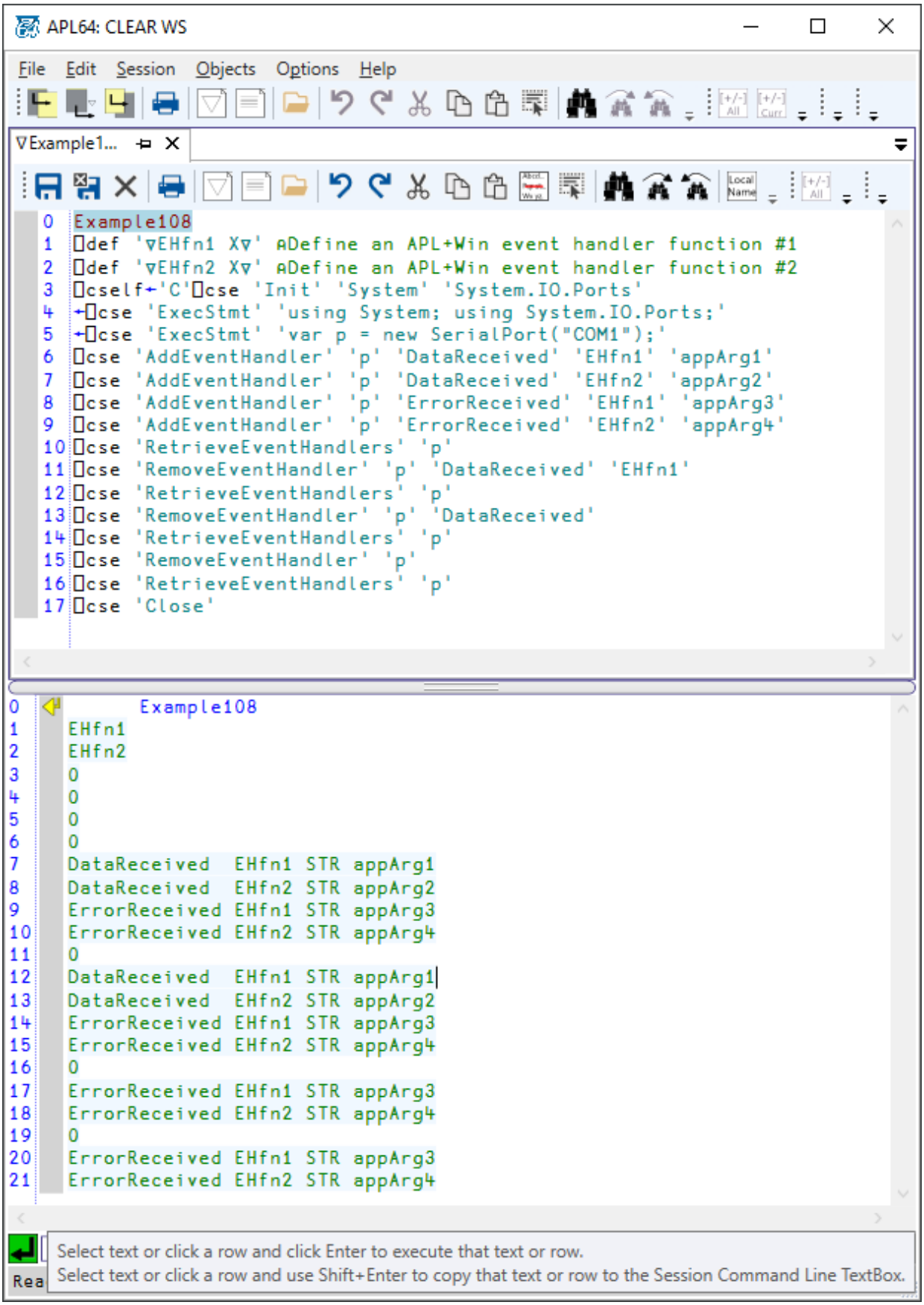
- Element #1 is required to contain the name of the C# object which exposed the event to which the APL64 event handler function is subscribed, e.g. 'myButton'
- Element #2 is optional and if present contains the name of the C# event exposed by the specified C# object, e.g. 'HelpButtonClicked'
- Element #3 is optional and if present contains the name of the APL64 event handler function subscribed to the C# event, e.g. 'EHfn'

- The action of the CSE 'RemoveEventHandler' method depends on the number of arguments provided to the method:
- If only argument element #1 is present, the action of the CSE 'RemoveEventHandler' method is to remove all the APL64 event handler subscriptions for the C# object specified in argument element #1
- If only argument elements #1 and #2 are present, the action of the CSE 'RemoveEventHandler' method is to remove all the APL64 event handler subscriptions for the C# object specified in argument element #1 which are for the C# event specified in argument element #2
- If all three argument elements are present, the action of the CSE 'RemoveEventHandler' method is to remove the APL64 event handler subscription for the specified C# object in argument element #1 which are for the C# event specified in argument element #2 which are for the APL64 event handler function specified in argument element #3.

Example #108:

This example illustrates all variations of the CSE ‘RemoveEventHandler’ method.

```
Example108
⎕def '∇EHfn1 X∇' ⍝Define an APL+Win event handler function #1
⎕def '∇EHfn2 X∇' ⍝Define an APL+Win event handler function #2
⎕cself←'C'⎕cse 'Init' 'System' 'System.IO.Ports'
←⎕cse 'ExecStmt' 'using System; using System.IO.Ports;'
←⎕cse 'ExecStmt' 'var p = new SerialPort("COM1");'
⎕cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn1' 'appArg1'
⎕cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn2' 'appArg2'
⎕cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn1' 'appArg3'
⎕cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn2' 'appArg4'
⎕cse 'RetrieveEventHandlers' 'p'
⎕cse 'RemoveEventHandler' 'p' 'DataReceived' 'EHfn1'
⎕cse 'RetrieveEventHandlers' 'p'
⎕cse 'RemoveEventHandler' 'p' 'DataReceived'
⎕cse 'RetrieveEventHandlers' 'p'
⎕cse 'RemoveEventHandler' 'p'
⎕cse 'RetrieveEventHandlers' 'p'
⎕cse 'Close'
```


APL64: CLEAR WS
File Edit Session Objects Options Help
∇Example1...
0 Example108
1 ⎕def '∇EHfn1 X∇' ⍝Define an APL+Win event handler function #1
2 ⎕def '∇EHfn2 X∇' ⍝Define an APL+Win event handler function #2
3 ⎕cself←'C'⎕cse 'Init' 'System' 'System.IO.Ports'
4 ←⎕cse 'ExecStmt' 'using System; using System.IO.Ports;'
5 ←⎕cse 'ExecStmt' 'var p = new SerialPort("COM1");'
6 ⎕cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn1' 'appArg1'
7 ⎕cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn2' 'appArg2'
8 ⎕cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn1' 'appArg3'
9 ⎕cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn2' 'appArg4'
10 ⎕cse 'RetrieveEventHandlers' 'p'
11 ⎕cse 'RemoveEventHandler' 'p' 'DataReceived' 'EHfn1'
12 ⎕cse 'RetrieveEventHandlers' 'p'
13 ⎕cse 'RemoveEventHandler' 'p' 'DataReceived'
14 ⎕cse 'RetrieveEventHandlers' 'p'
15 ⎕cse 'RemoveEventHandler' 'p'
16 ⎕cse 'RetrieveEventHandlers' 'p'
17 ⎕cse 'Close'
0 Example108
1 EHfn1
2 EHfn2
3 0
4 0
5 0
6 0
7 DataReceived EHfn1 STR appArg1
8 DataReceived EHfn2 STR appArg2
9 ErrorReceived EHfn1 STR appArg3
10 ErrorReceived EHfn2 STR appArg4
11 0
12 DataReceived EHfn1 STR appArg1
13 DataReceived EHfn2 STR appArg2
14 ErrorReceived EHfn1 STR appArg3
15 ErrorReceived EHfn2 STR appArg4
16 0
17 ErrorReceived EHfn1 STR appArg3
18 ErrorReceived EHfn2 STR appArg4
19 0
20 ErrorReceived EHfn1 STR appArg3
21 ErrorReceived EHfn2 STR appArg4
Select text or click a row and click Enter to execute that text or row.
Select text or click a row and use Shift+Enter to copy that text or row to the Session Command Line TextBox.
Rea

CSE RetrieveCustomEventHandlers Method

This method will return an APL64 array with one row for each APL64 custom event handler function subscribed to a .Net object event.

The number of columns in the result depends on the number of elements in the right argument of the CSE 'RetrieveCustomEventHandlers' method.

- If both the 1st and 2nd elements of the right argument are present, the columns of the result are:
  - Name of the APL64 custom event handler function o Type of subscription, i.e. 'STR' if the event subscription is due to the CSE 'AddCustomEventHandler' method or 'EXE' if the event subscription is due to the CSE 'AddCustomEventHandlerEx' method.
  - The programmer-supplied text vector representing the application-specific argument to the APL64 custom event handler function specified when the CSE 'AddCustomEventHandler' or 'AddCustomEventHandlerEx' method was used to subscribe to this event.
- If only the 1st element of the right argument is present, the columns of the result are
  - Name of the C# custom event which has been subscribed
  - Name of the APL64 custom event handler function
  - Type of subscription, i.e. 'STR' if the event subscription is due to the CSE 'AddCustomEventHandler' method or 'EXE' if the event subscription is due to the CSE 'AddCustomEventHandlerEx' method.
  - Application-specific argument to the APL64 event handler function specified when the CSE 'AddCustomEventHandler' or 'AddCustomEventHandlerEx' method was used to subscribe to this event.

The right argument of the CSE 'RetrieveCustomEventHandlers' method is a vector of up to two APL64 text vectors:

- Element #1 is required and contains the name of the C# object which exposes the custom event to which APL64 custom event handlers functions have been subscribed using the CSE 'AddCustomEventHandler' or 'AddCustomEventHandlerEx' methods.
- Element #2 is optional and if present contains the name of the C# custom event which is exposed by the specified C# object.

CSE RetrieveEventHandlers Method

This method will return an APL64 array with one row for each APL64 event handler function subscribed to a .Net object event.

The number of columns in the result depends on the number of elements in the right argument of the CSE 'RetrieveEventHandlers' method.

- If both the 1st and 2nd elements of the right argument are present, the columns of the result are:
  - Name of the APL64 event handler function
  - Type of subscription, i.e. 'STR' if the event subscription is due to the CSE 'AddEventHandler' method or 'EXE' if the event subscription is due to the CSE 'AddEventHandlerEx' method.

- The programmer-supplied text vector representing the application-specific argument to the APL64 event handler function specified when the CSE 'AddEventHandler' or 'AddEventHandlerEx' method was used to subscribe to this event.

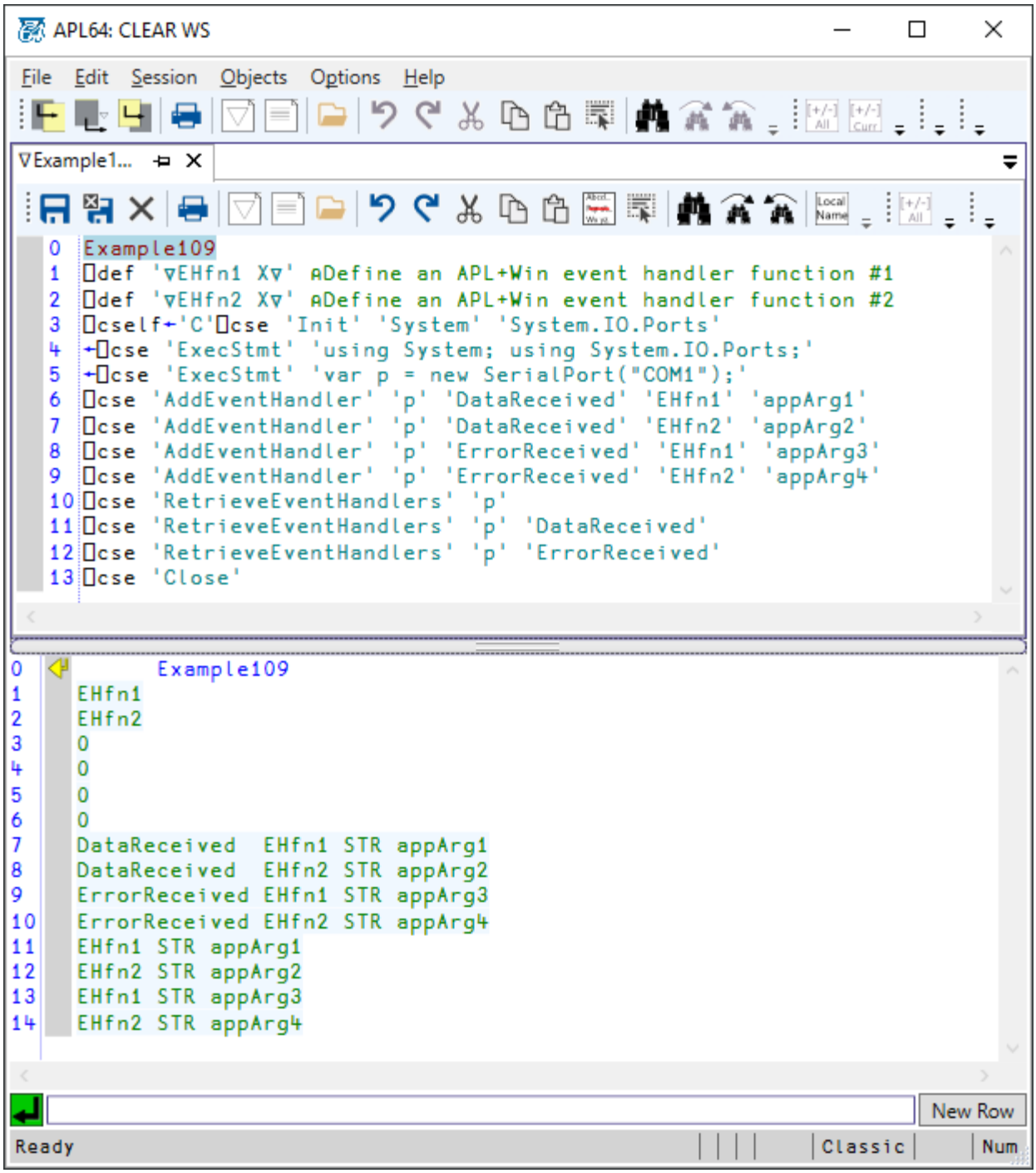
- If only the 1st element of the right argument is present, the columns of the result are:
  - Name of the C# event which has been subscribed
  - Name of the APL64 event handler function
  - Type of subscription, i.e. 'STR' if the event subscription is due to the CSE 'AddEventHandler' method or 'EXE' if the event subscription is due to the CSE 'AddEventHandlerEx' method.
  - Application-specific argument to the APL64 event handler function specified when the CSE 'AddEventHandler' or 'AddEventHandlerEx' method was used to subscribe to this event.

The right argument of the CSE 'RetrieveEventHandlers' method is a vector of up to two APL64 text vectors:

- Element #1 is required and contains the name of the C# object which exposes the event to which APL64 event handlers functions have been subscribed using the CSE 'AddEventHandler' method.
- Element #2 is optional and if present contains the name of the C# event which is exposed by the specified C# object.

Example #109:
This example illustrates all variations of the CSE 'RetrieveEventhHandlers' method.

```
Example109
□def '∇EHfn1 X∇' ⍝Define an APL+Win event handler function #1
□def '∇EHfn2 X∇' ⍝Define an APL+Win event handler function #2
□cself←'C'□cse 'Init' 'System' 'System.IO.Ports'
←□cse 'ExecStmt' 'using System; using System.IO.Ports;'
←□cse 'ExecStmt' 'var p = new SerialPort("COM1");'
□cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn1' 'appArg1'
□cse 'AddEventHandler' 'p' 'DataReceived' 'EHfn2' 'appArg2'
□cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn1' 'appArg3'
□cse 'AddEventHandler' 'p' 'ErrorReceived' 'EHfn2' 'appArg4'
□cse 'RetrieveEventHandlers' 'p'
□cse 'RetrieveEventHandlers' 'p' 'DataReceived'
□cse 'RetrieveEventHandlers' 'p' 'ErrorReceived'
□cse 'Close'
```



CSE SetValue Method

The CSE 'SetValue' method is used to assign a value from an object in the APL64 workspace to a C# object in the instance of a CSE object. The action of the CSE 'SetValue' method is affected by the value of the CSE 'returnonerror' property.

There are two required right argument elements to the 'SetValue' method. The first right argument element is a text vector containing the name of the pre-existing, target C# object. The second right

argument element is either the name of the source APL64 object or the source data represented as an APL64 executable expression.

Use the CSE 'GetLastError' method to obtain the text of the error message if the CSE 'SetValue' method fails.

Prior to using the CSE SetValue method on a C# variable, it is necessary to declare the C# variable's name and data type using CSE 'Exec', 'ExecStmt' or 'ExecFile' methods.

The data type of the source APL object must be conformable with the data type of the target C# object or the CSE will throw an exception. For numeric data types conversion is automatic in the CSE when there is no potential loss of precision, e.g. Int16 to Int32 or Int32 to Double.

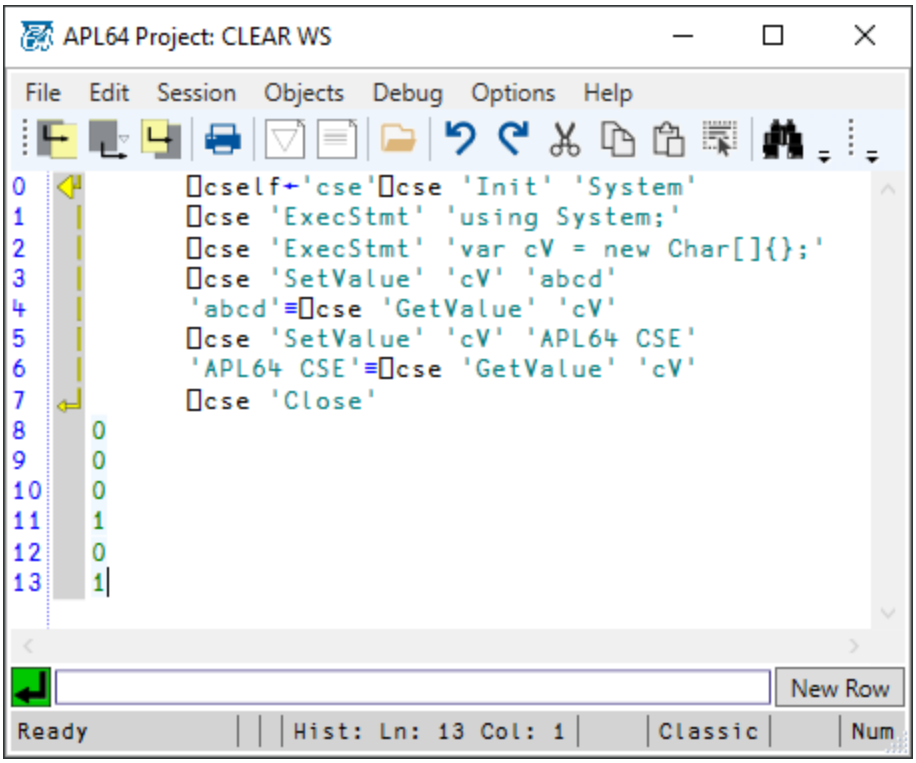
The shape of the source APL object must also be conformable with the shape of the target C# object or the CSE will throw an exception. For arrays of any shape with values which are APL64 data types conversion is automatic in the CSE.

When the type of the APL64 variable is not conformable with the type of the target C# variable, several approaches are possible:

- Coerce APL64 to provide the appropriate data type to the C# target property, or
- Create an intermediate C# variable of the appropriate data type and then use an explicit C# conversion or cast to convert the intermediate variable's value in order to set the target C# variable properly.
- Create a C# 'extension' method to handle the desired coercion.

Example #5
Set the value of a .Net character vector from an APL64 character vector.

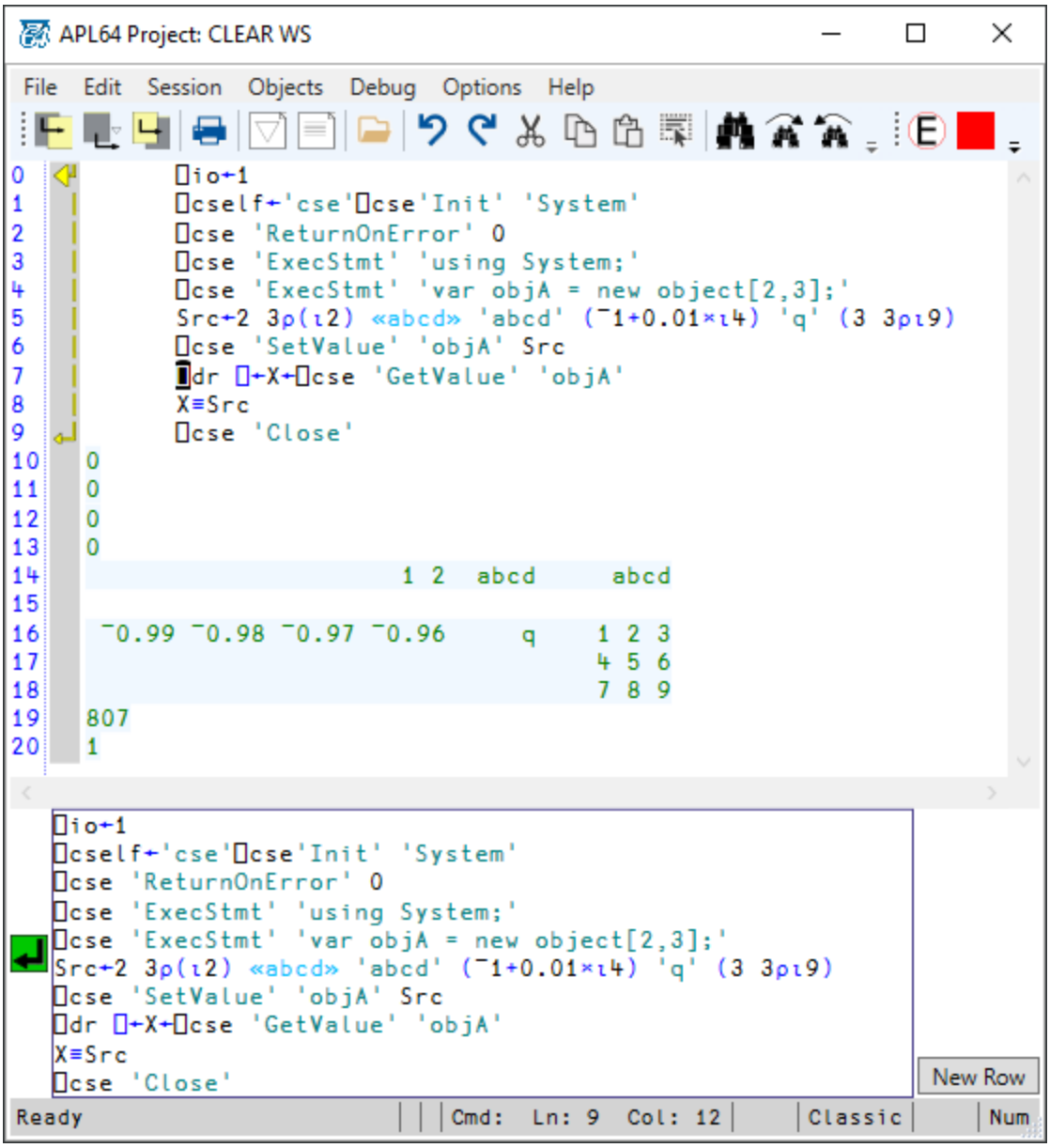
```
⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'var cV = new Char[]{};'
⎕cse 'SetValue' 'cV' 'abcd'
'abcd'≡⎕cse 'GetValue' 'cV'
⎕cse 'SetValue' 'cV' 'APL64 CSE'
'APL64 CSE'≡⎕cse 'GetValue' 'cV'
⎕cse 'Close'
```



Example #182

Set Value of Object Array from Nested APL64 Array

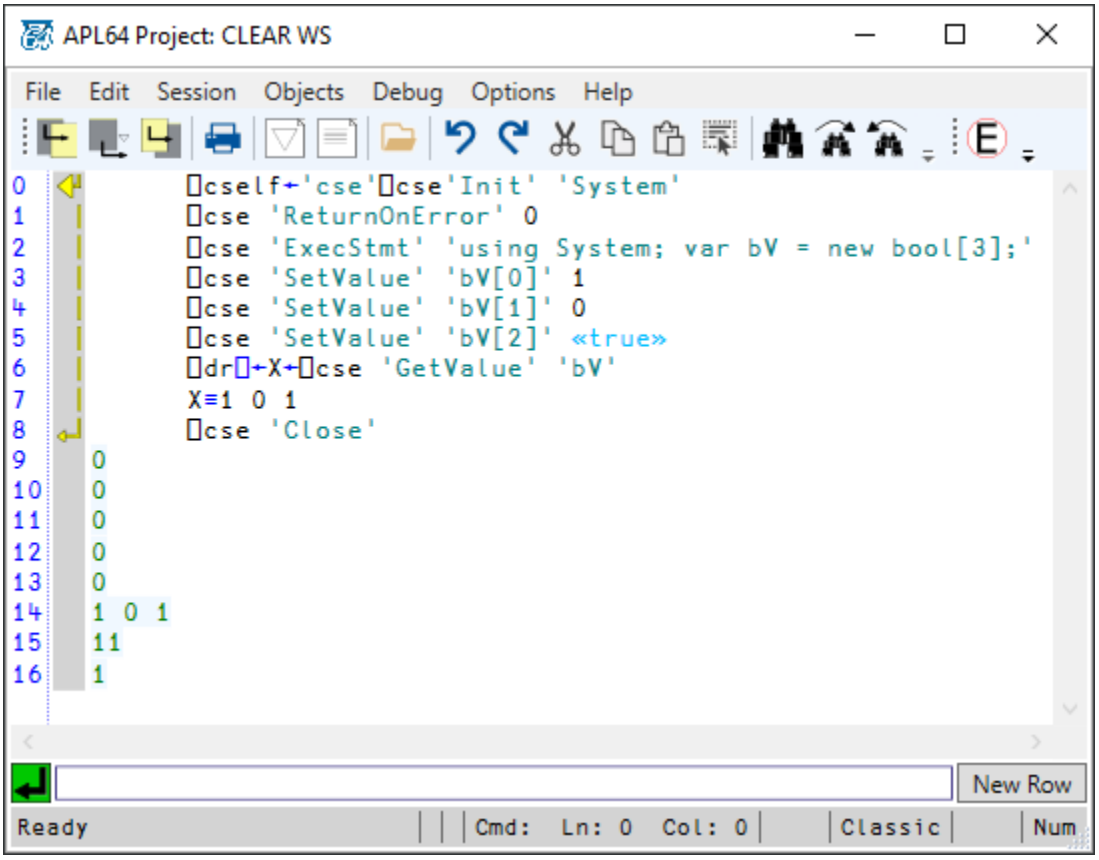
```
⎕io←1
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'var objA = new object[2,3];'
Src←2 3⍴(⍳2) «abcd» 'abcd' (¯1+0.01×⍳4) 'q' (3 3⍴⍳9)
⎕cse 'SetValue' 'objA' Src
⎕dr ⎕←X←⎕cse 'GetValue' 'objA'
X≡Src
⎕cse 'Close'
```



Example #211

Set Value of Boolean Homogeneous Vector

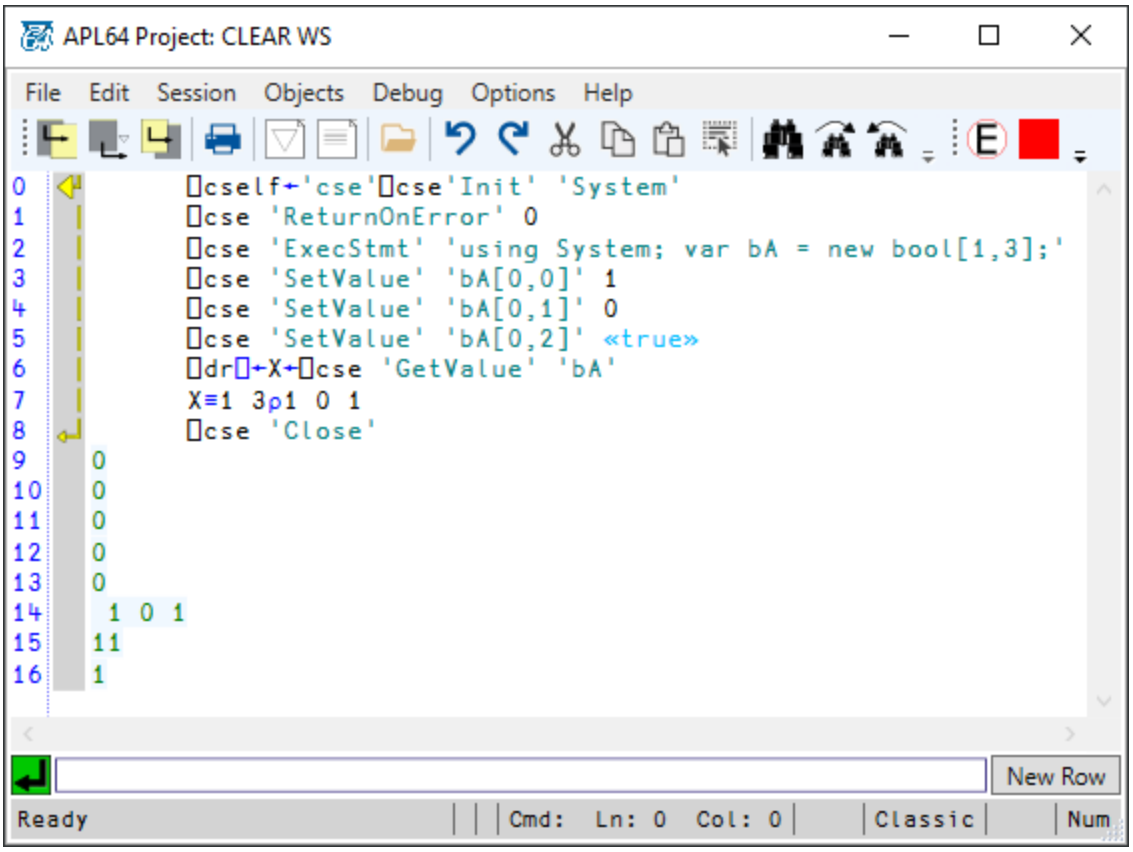
```
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System; var bV = new bool[3];'
⎕cse 'SetValue' 'bV[0]' 1
⎕cse 'SetValue' 'bV[1]' 0
⎕cse 'SetValue' 'bV[2]' «true»
⎕dr⎕←X←⎕cse 'GetValue' 'bV'
X≡1 0 1
⎕cse 'Close'
```



Example #212

Set Value of Boolean Homogeneous Matrix

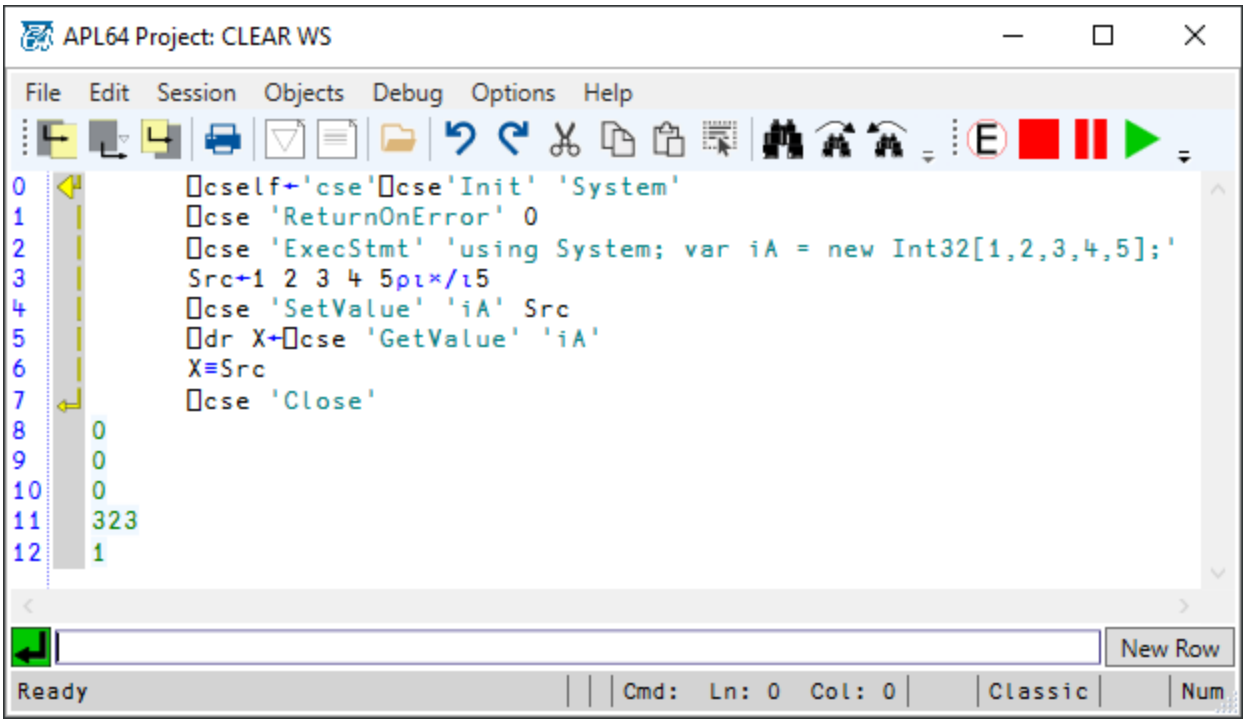
```
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System; var bA = new bool[1,3];'
⎕cse 'SetValue' 'bA[0,0]' 1
⎕cse 'SetValue' 'bA[0,1]' 0
⎕cse 'SetValue' 'bA[0,2]' «true»
⎕dr⎕←X←⎕cse 'GetValue' 'bA'
X≡1 3⍴1 0 1
⎕cse 'Close'
```



Example #213

Set Value of Integer Homogeneous Array

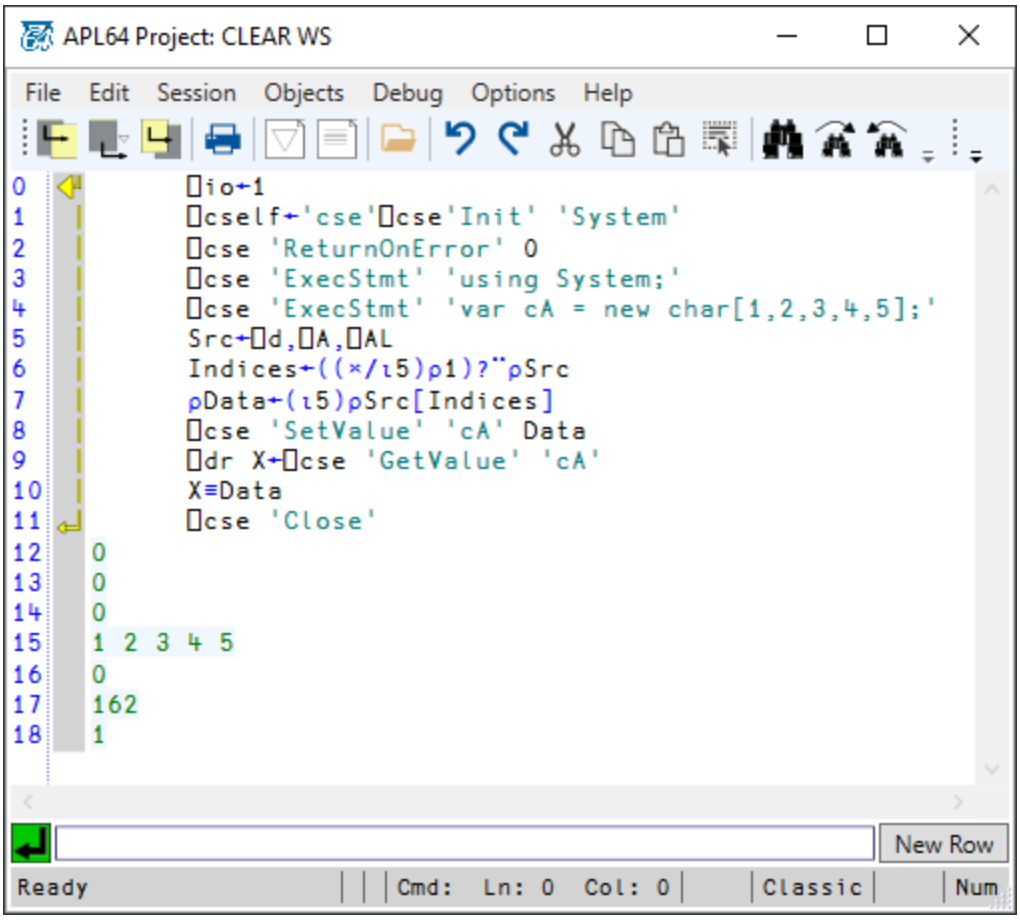
```
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System; var iA = new Int32[1,2,3,4,5];'
Src←1 2 3 4 5⍴⍳×/⍳5
⎕cse 'SetValue' 'iA' Src
⎕dr X←⎕cse 'GetValue' 'iA'
X≡Src
⎕cse 'Close'
```



Example #215

Set Value of Character Homogeneous Array

```
⎕io←1
⎕cself←'cse'⎕cse'Init' 'System'
⎕cse 'ReturnOnError' 0
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'var cA = new char[1,2,3,4,5];'
Src←⎕d,⎕A,⎕AL
Indices←((×/⍳5)⍴1)?¨⍴Src
⍴Data←(⍳5)⍴Src[Indices]
⎕cse 'SetValue' 'cA' Data
⎕dr X←⎕cse 'GetValue' 'cA'
X≡Data
⎕cse 'Close'
```

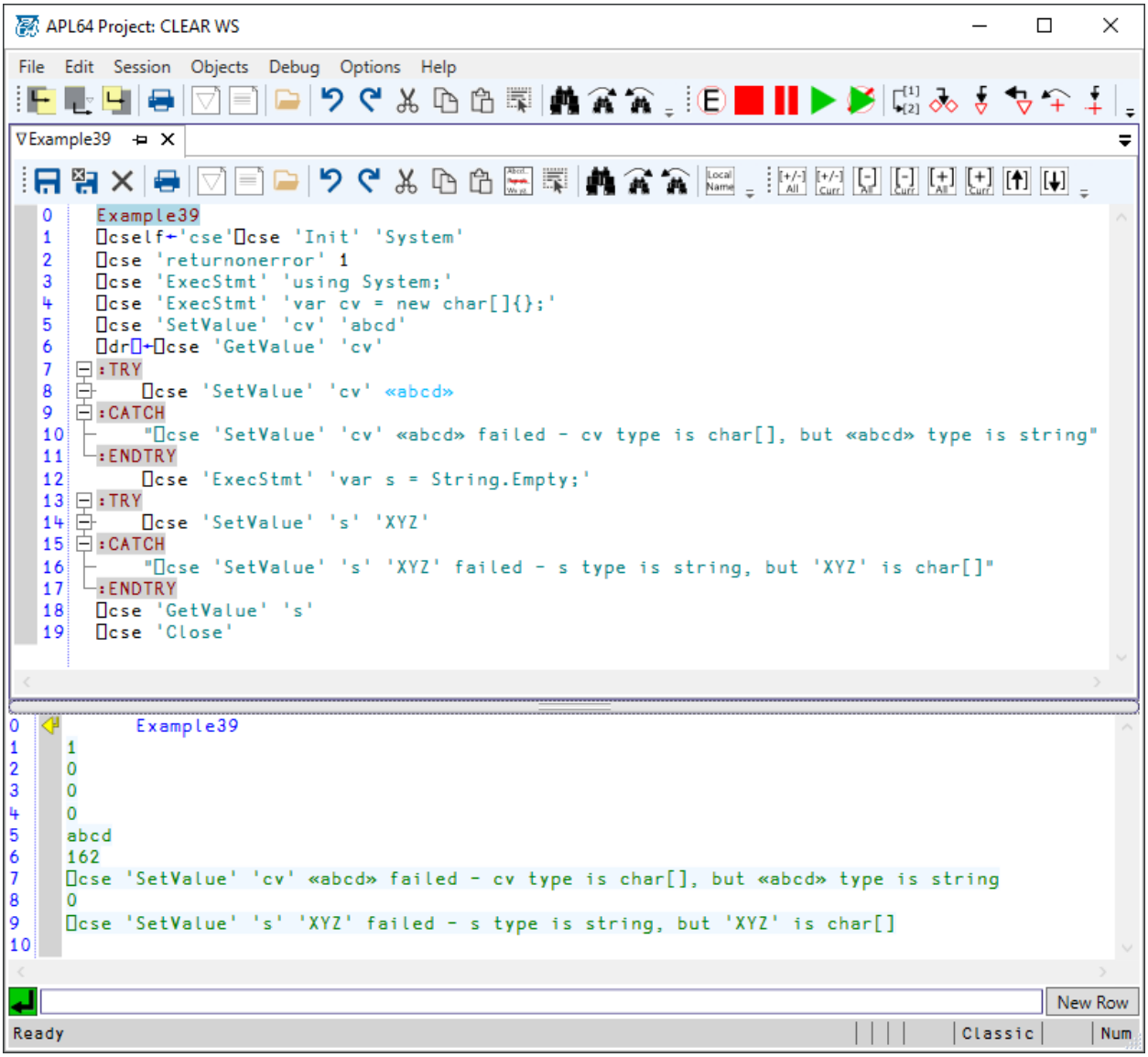


Example #39

This example illustrates using the CSE SetValue method to provide values for C# String and Char[] data types.

```
Example39
⎕cself←'cse'⎕cse 'Init' 'System'
⎕cse 'returnonerror' 1
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'var cv = new char[]{};'
⎕cse 'SetValue' 'cv' 'abcd'
⎕dr⎕←⎕cse 'GetValue' 'cv'
:TRY
   ⎕cse 'SetValue' 'cv' «abcd»
:CATCH
   "⎕cse 'SetValue' 'cv' «abcd» failed - cv type is char[], but «abcd» type is string"
:ENDTRY
   ⎕cse 'ExecStmt' 'var s = String.Empty;'
:TRY
```

```
    cse 'SetValue' 's' 'XYZ'
:CATCH
    "cse 'SetValue' 's' 'XYZ' failed - s type is string, but 'XYZ' is char[]"
:ENDTRY
cse 'GetValue' 's'
cse 'Close'
```

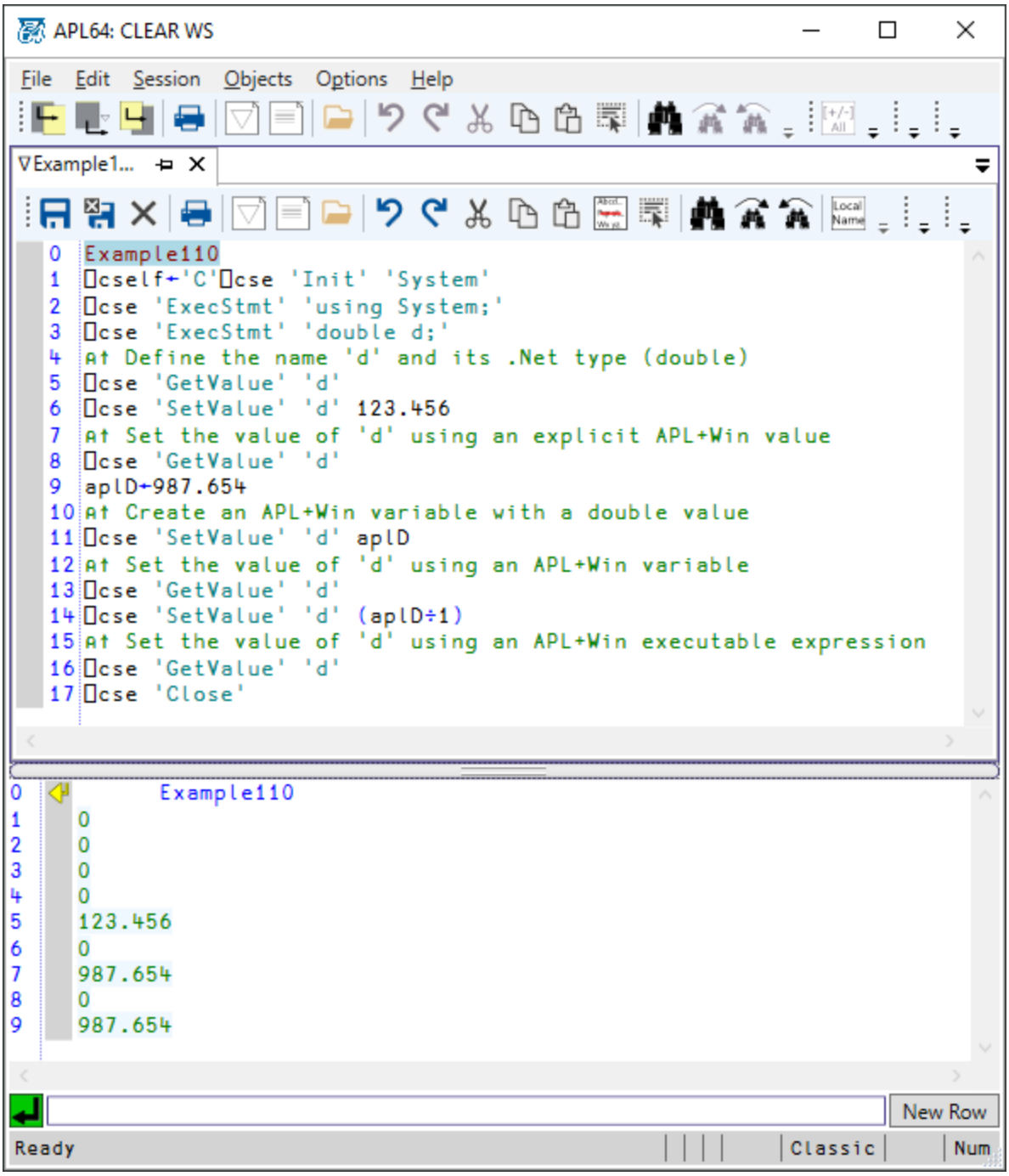


Example #110:

In this example the target C# variable has been defined by name and .Net type. The CSE 'SetValue' method is used to set the value of this C# variable using:

- An explicit APL64 value
- An APL64 variable
- An APL64 executable expression

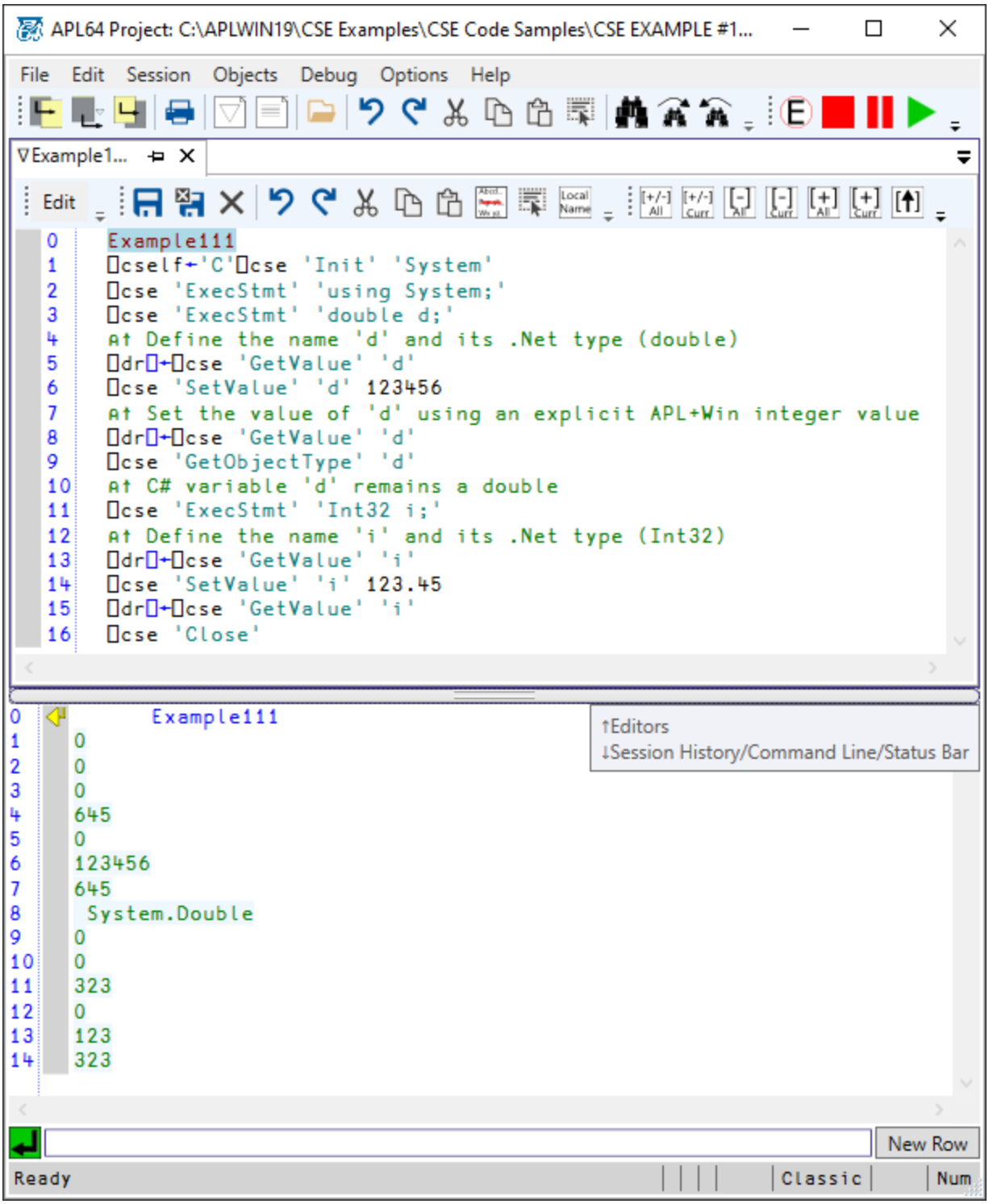
```
Example110
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'double d;'
⍝↑ Define the name 'd' and its .Net type (double)
⎕cse 'GetValue' 'd'
⎕cse 'SetValue' 'd' 123.456
⍝↑ Set the value of 'd' using an explicit APL+Win value
⎕cse 'GetValue' 'd'
aplD←987.654
⍝↑ Create an APL+Win variable with a double value
⎕cse 'SetValue' 'd' aplD
⍝↑ Set the value of 'd' using an APL+Win variable
⎕cse 'GetValue' 'd'
⎕cse 'SetValue' 'd' (aplD÷1)
⍝↑ Set the value of 'd' using an APL+Win executable expression
⎕cse 'GetValue' 'd'
⎕cse 'Close'
```



Example #111:

In this example the CSE 'SetValue' method is used to set the value of a C# variable 'd' of type double using an APL64 integer value. C# implicitly coerces the APL64 integer type to the C# double type. Also, in this example the CSE 'SetValue' method is used to set the value of a C# variable 'I' of type Int32 using an APL64 double value. The CSE implicitly coerces the APL64 double type to the C# Int32 type, with an associated loss of precision. This illustrates the need to carefully prepare the APL64 source and C# target variables when using the CSE 'SetValue' method.

```
Example111
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'double d;'
⍝↑ Define the name 'd' and its .Net type (double)
⎕dr⎕←⎕cse 'GetValue' 'd'
⎕cse 'SetValue' 'd' 123456
⍝↑ Set the value of 'd' using an explicit APL+Win integer value
⎕dr⎕←⎕cse 'GetValue' 'd'
⎕cse 'GetObjectType' 'd'
⍝↑ C# variable 'd' remains a double
⎕cse 'ExecStmt' 'Int32 i;'
⍝↑ Define the name 'i' and its .Net type (Int32)
⎕dr⎕←⎕cse 'GetValue' 'i'
⎕cse 'SetValue' 'i' 123.45
⎕dr⎕←⎕cse 'GetValue' 'i'
⎕cse 'Close'
```

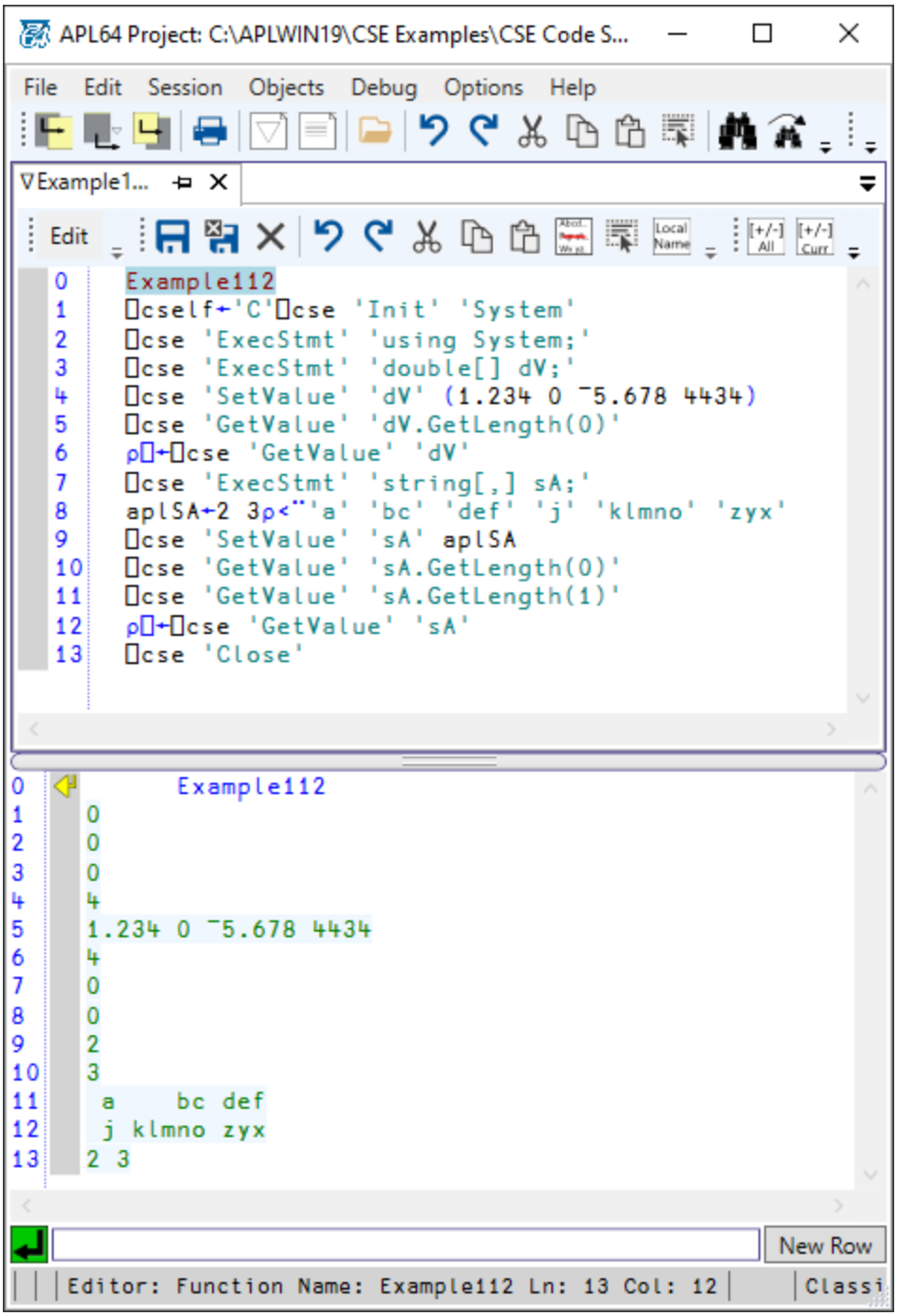


Example #112

In this example below, C# arrays are defined and their values are set from APL64 using the CSE 'SetValue' method. Notice that the APL64 numeric vector containing doubles is conformable with the C# variable 'dV' of type double[].

Example112

```
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'double[] dV;'
⎕cse 'SetValue' 'dV' (1.234 0 ¯5.678 4434)
⎕cse 'GetValue' 'dV.GetLength(0)'
ρ⎕←⎕cse 'GetValue' 'dV'
⎕cse 'ExecStmt' 'string[,] sA;'
aplSA←2 3ρ<¨'a' 'bc' 'def' 'j' 'klmno' 'zyx'
⎕cse 'SetValue' 'sA' aplSA
⎕cse 'GetValue' 'sA.GetLength(0)'
⎕cse 'GetValue' 'sA.GetLength(1)'
ρ⎕←⎕cse 'GetValue' 'sA'
⎕cse 'Close'
```

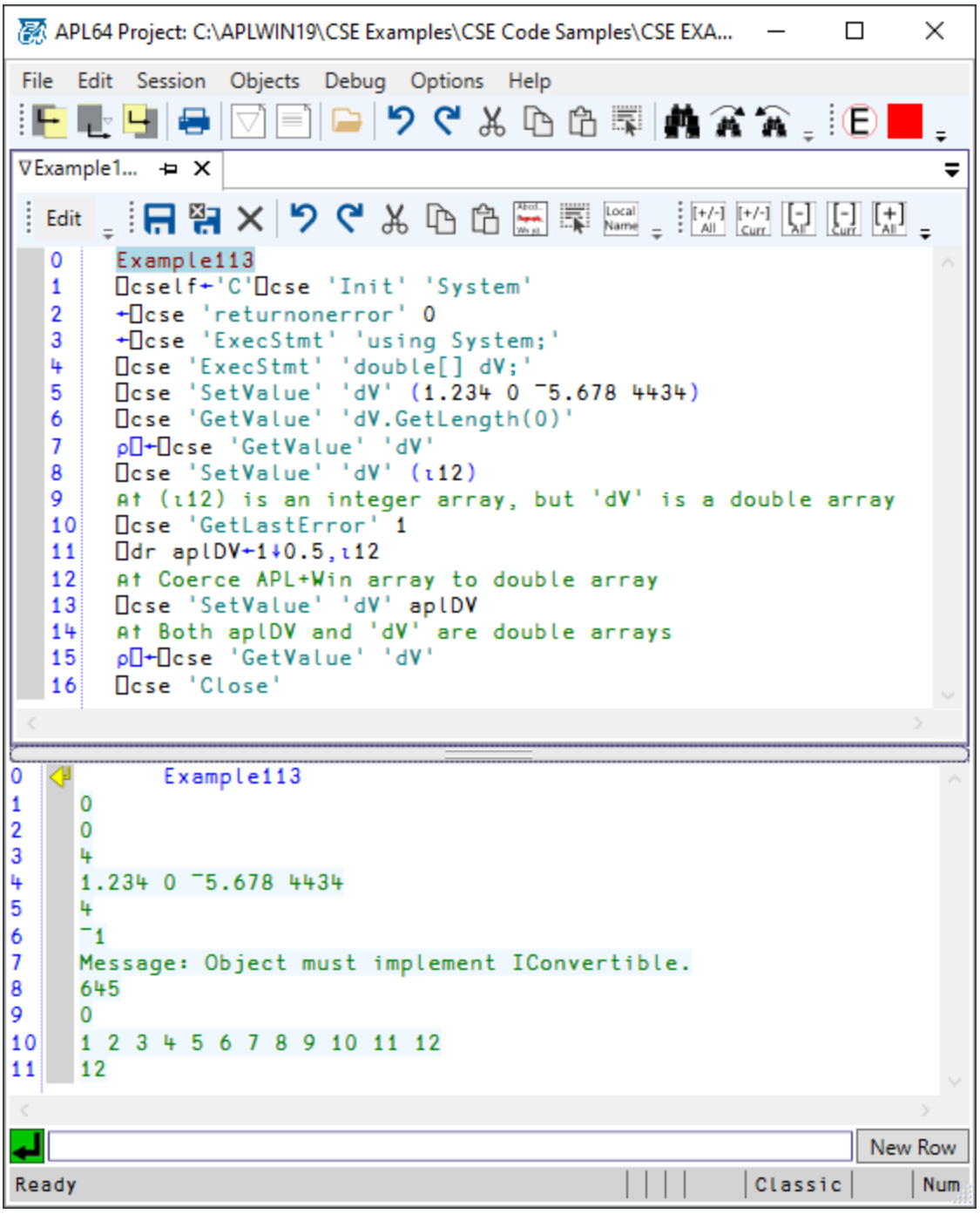


Example #113

In this example a C# exception is thrown when APL64 attempts to set the values of a C# array of type double[] using an APL64 array of type integer. Since a scalar type is not involved, the CSE does not perform an implicit coercion from the APL64 integer array to the C# double array. This example illustrates a method to coerce APL64 to provide the conformable double array to the CSE 'SetValue' method.

```
Example113
⎕cself←'C'⎕cse 'Init' 'System'
```

```
←⎕cse 'returnonerror' 0
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'double[] dV;'
⎕cse 'SetValue' 'dV' (1.234 0 ¯5.678 4434)
⎕cse 'GetValue' 'dV.GetLength(0)'
ρ⎕←⎕cse 'GetValue' 'dV'
⎕cse 'SetValue' 'dV' (⍳12)
⍝↑ (⍳12) is an integer array, but 'dV' is a double array
⎕cse 'GetLastError' 1
⎕dr aplDV←1↓0.5,⍳12
⍝↑ Coerce APL+Win array to double array
⎕cse 'SetValue' 'dV' aplDV
⍝↑ Both aplDV and 'dV' are double arrays
ρ⎕←⎕cse 'GetValue' 'dV'
⎕cse 'Close'
```

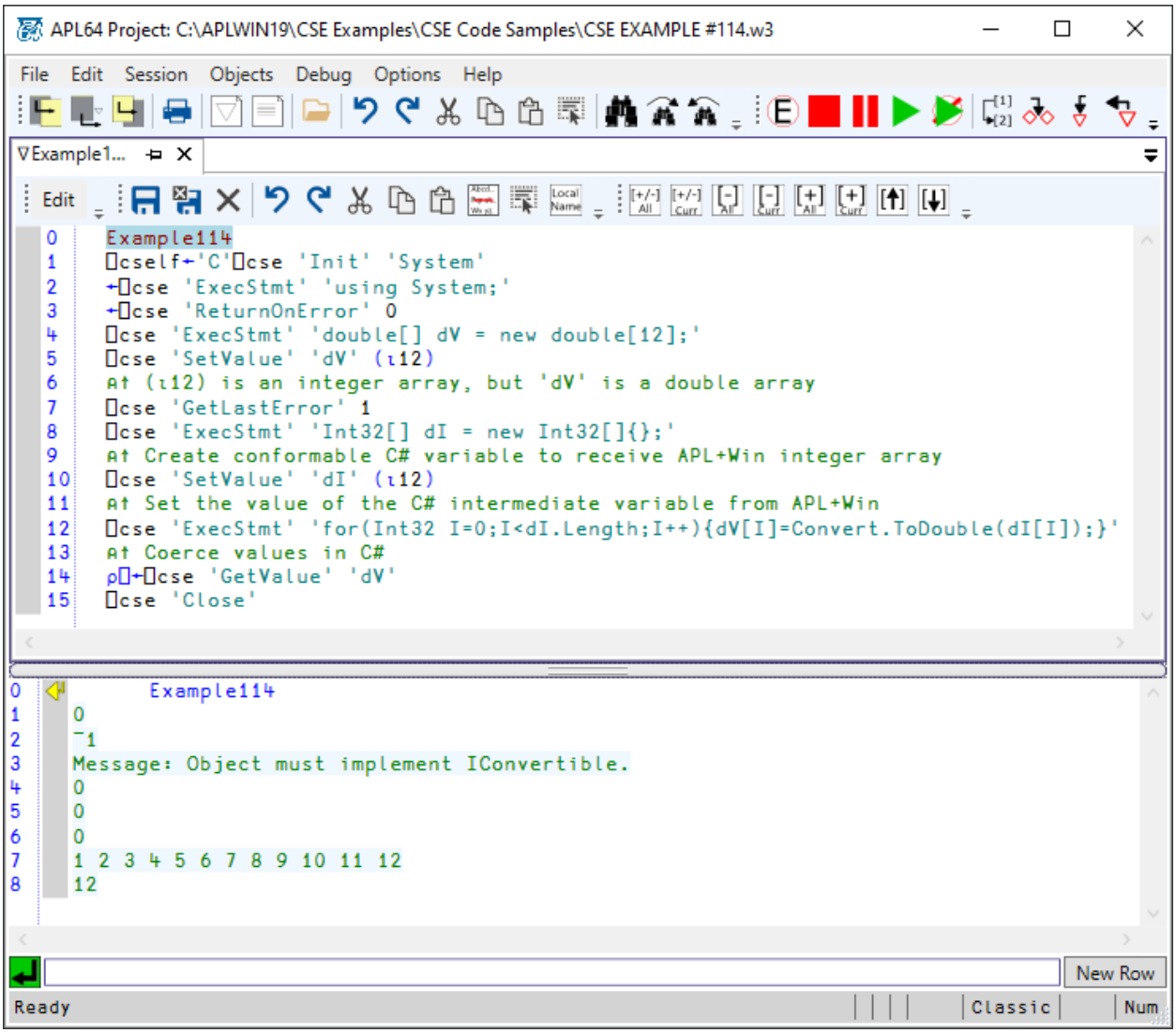


Example #114

This example is an analogue of Example #113, except that the data type coercion from Int32[] to double[] is performed in C# rather than in APL64.

```
Example114
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
←⎕cse 'ReturnOnError' 0
```

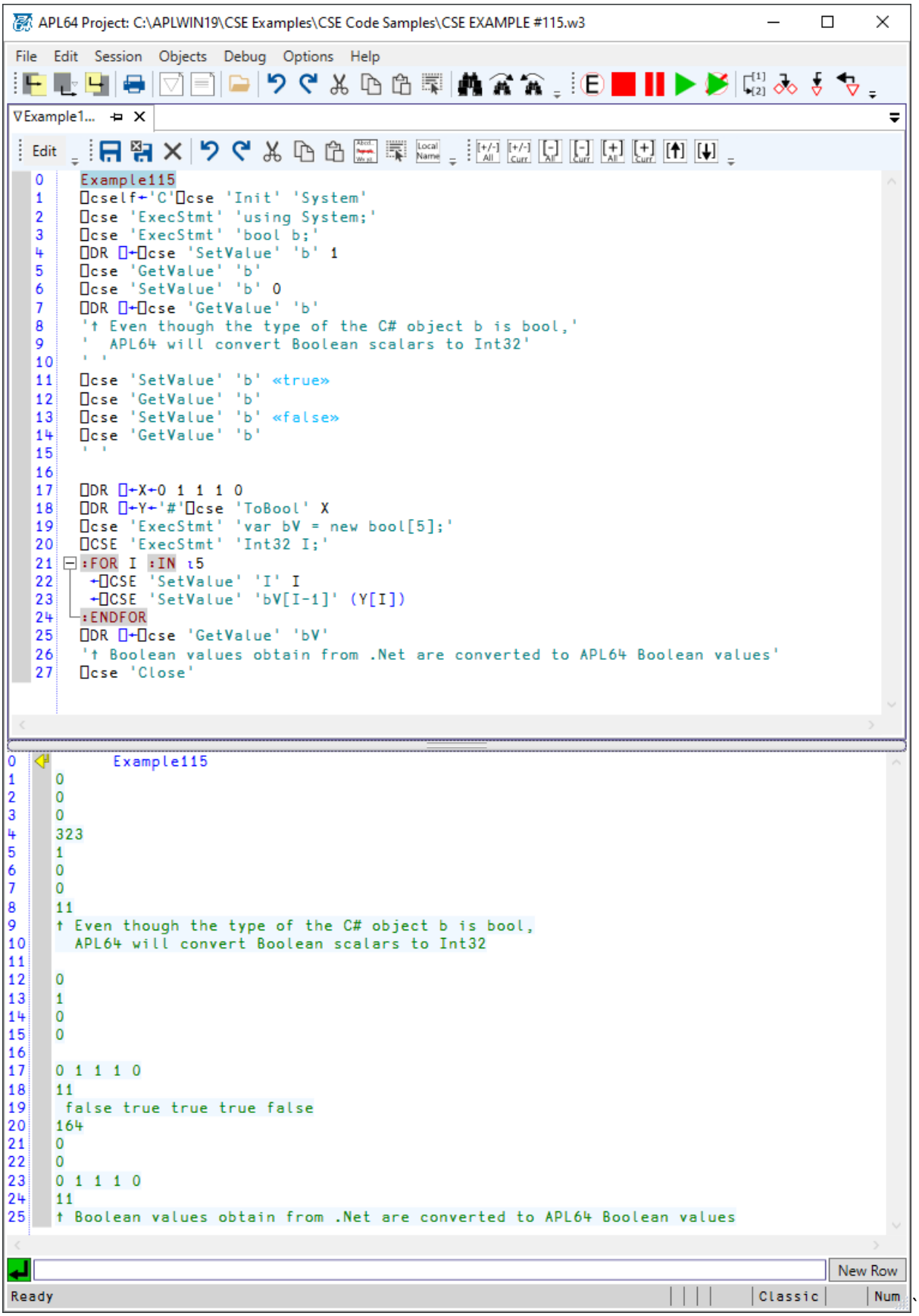
```
⎕cse 'ExecStmt' 'double[] dV = new double[12];'
⎕cse 'SetValue' 'dV' (⍳12)
⍝↑ (⍳12) is an integer array, but 'dV' is a double array
⎕cse 'GetLastError' 1
⎕cse 'ExecStmt' 'Int32[] dI = new Int32[]{};'
⍝↑ Create conformable C# variable to receive APL+Win integer array
⎕cse 'SetValue' 'dI' (⍳12)
⍝↑ Set the value of the C# intermediate variable from APL+Win
⎕cse 'ExecStmt' 'for(Int32 I=0;I<dI.Length;I++){dV[I]=Convert.ToDouble(dI[I]);}'
⍝↑ Coerce values in C#
⍴⎕←⎕cse 'GetValue' 'dV'
⎕cse 'Close'
```



Example #115:

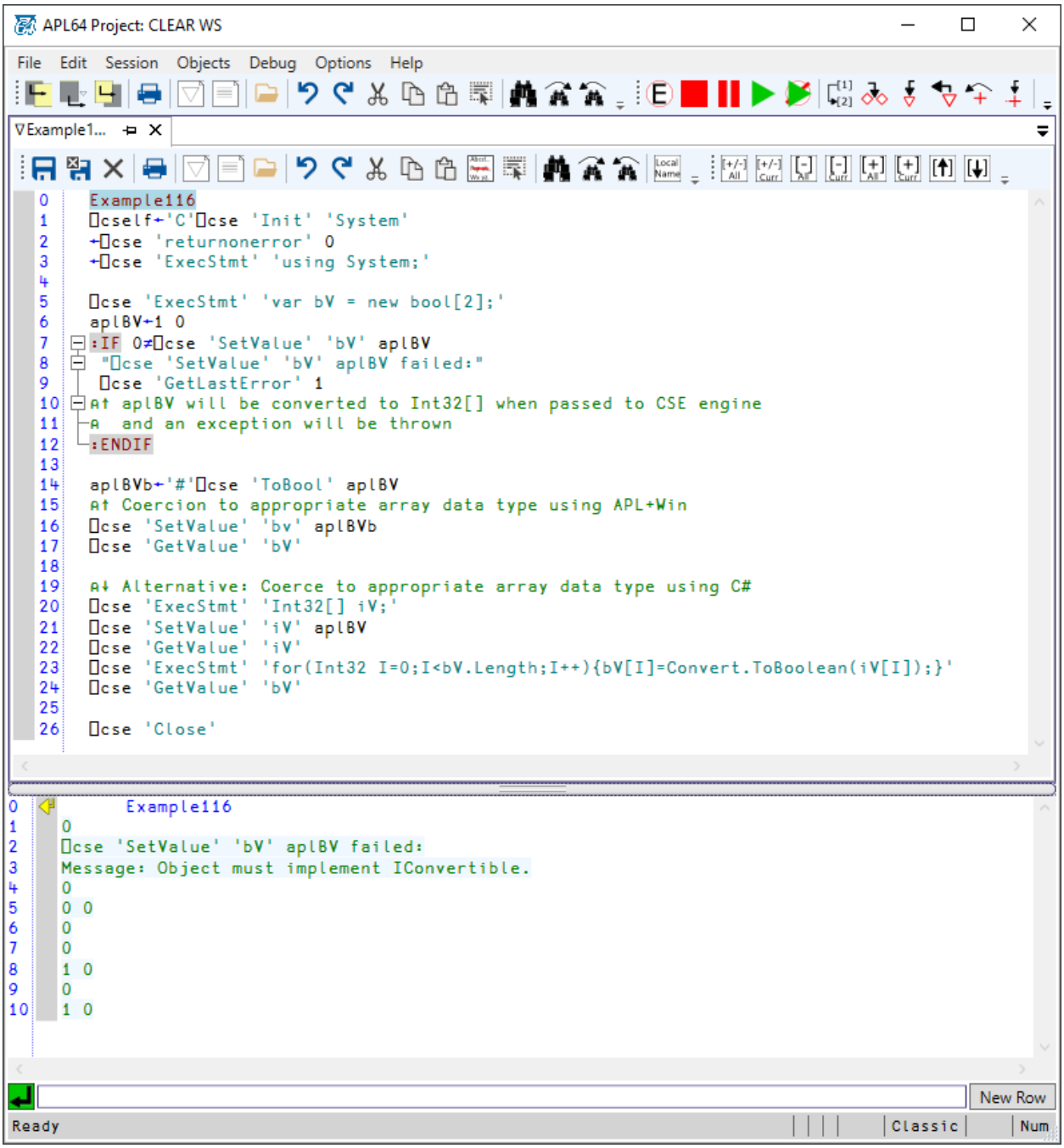
- APL64 Boolean objects are converted to Int32 .Net objects so that numeric operations are properly carried out.
- To set the value of a .Net Boolean object, provide an APL64 string value of «true» or «false»
- The CSE object method ToBool may be helpful when complex C# objects containing Boolean values are involved

```
Example115
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'bool b;'
□DR □←□cse 'SetValue' 'b' 1
□cse 'GetValue' 'b'
□cse 'SetValue' 'b' 0
□DR □←□cse 'GetValue' 'b'
'↑ Even though the type of the C# object b is bool,'
'  APL64 will convert Boolean scalars to Int32'
' '
□cse 'SetValue' 'b' «true»
□cse 'GetValue' 'b'
□cse 'SetValue' 'b' «false»
□cse 'GetValue' 'b'
' '

□DR □←X←0 1 1 1 0
□DR □←Y←'#'□cse 'ToBool' X
□cse 'ExecStmt' 'var bV = new bool[5];'
□CSE 'ExecStmt' 'Int32 I;'
:FOR I :IN ⍳5
 ←□CSE 'SetValue' 'I' I
 ←□CSE 'SetValue' 'bV[I-1]' (Y[I])
:ENDFOR
□DR □←□cse 'GetValue' 'bV'
'↑ Boolean values obtain from .Net are converted to APL64 Boolean values'
□cse 'Close'
```


APL64 Project: C:\APLWIN19\CSE Examples\CSE Code Samples\CSE EXAMPLE #115.w3
File Edit Session Objects Debug Options Help
Example1...
Edit
0 Example115
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'ExecStmt' 'using System;'
3 ⎕cse 'ExecStmt' 'bool b;'
4 ⎕DR ⍬←⎕cse 'SetValue' 'b' 1
5 ⎕cse 'GetValue' 'b'
6 ⎕cse 'SetValue' 'b' 0
7 ⎕DR ⍬←⎕cse 'GetValue' 'b'
8 '⍝ Even though the type of the C# object b is bool,'
9 ' APL64 will convert Boolean scalars to Int32'
10 ' '
11 ⎕cse 'SetValue' 'b' «true»
12 ⎕cse 'GetValue' 'b'
13 ⎕cse 'SetValue' 'b' «false»
14 ⎕cse 'GetValue' 'b'
15 ' '
16
17 ⎕DR ⍬←X←0 1 1 1 0
18 ⎕DR ⍬←Y←'#'⎕cse 'ToBool' X
19 ⎕cse 'ExecStmt' 'var bV = new bool[5];'
20 ⎕CSE 'ExecStmt' 'Int32 I;'
21 :FOR I :IN ⍳5
22 ⍝⎕CSE 'SetValue' 'I' I
23 ⍝⎕CSE 'SetValue' 'bV[I-1]' (Y[I])
24 :ENDFOR
25 ⎕DR ⍬←⎕cse 'GetValue' 'bV'
26 '⍝ Boolean values obtain from .Net are converted to APL64 Boolean values'
27 ⎕cse 'Close'
0 Example115
1 0
2 0
3 0
4 323
5 1
6 0
7 0
8 11
9 ⍝ Even though the type of the C# object b is bool,
10 APL64 will convert Boolean scalars to Int32
11
12 0
13 1
14 0
15 0
16
17 0 1 1 1 0
18 11
19 false true true true false
20 164
21 0
22 0
23 0 1 1 1 0
24 11
25 ⍝ Boolean values obtain from .Net are converted to APL64 Boolean values
New Row
Ready
Classic
Num

Example #116:
In this example the value of a C# Boolean array is set using the CSE ‘SetValue’ method from APL64. Explicit coercion of the APL64 values to conformable C# values is illustrated using APL64 and using C#.

```
Example116
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'returnonerror' 0
←⎕cse 'ExecStmt' 'using System;'

⎕cse 'ExecStmt' 'var bV = new bool[2];'
aplBV←1 0
:IF 0≠⎕cse 'SetValue' 'bV' aplBV
 "⎕cse 'SetValue' 'bV' aplBV failed:"
 ⎕cse 'GetLastError' 1
⍝↑ aplBV will be converted to Int32[] when passed to CSE engine
⍝  and an exception will be thrown
:ENDIF

aplBVb←'#'⎕cse 'ToBool' aplBV
⍝↑ Coercion to appropriate array data type using APL+Win
⎕cse 'SetValue' 'bv' aplBVb
⎕cse 'GetValue' 'bV'

⍝↓ Alternative: Coerce to appropriate array data type using C#
⎕cse 'ExecStmt' 'Int32[] iV;'
⎕cse 'SetValue' 'iV' aplBV
⎕cse 'GetValue' 'iV'
⎕cse 'ExecStmt' 'for(Int32 I=0;I<bV.Length;I++){bV[I]=Convert.ToBoolean(iV[I]);}'
⎕cse 'GetValue' 'bV'

⎕cse 'Close'
```

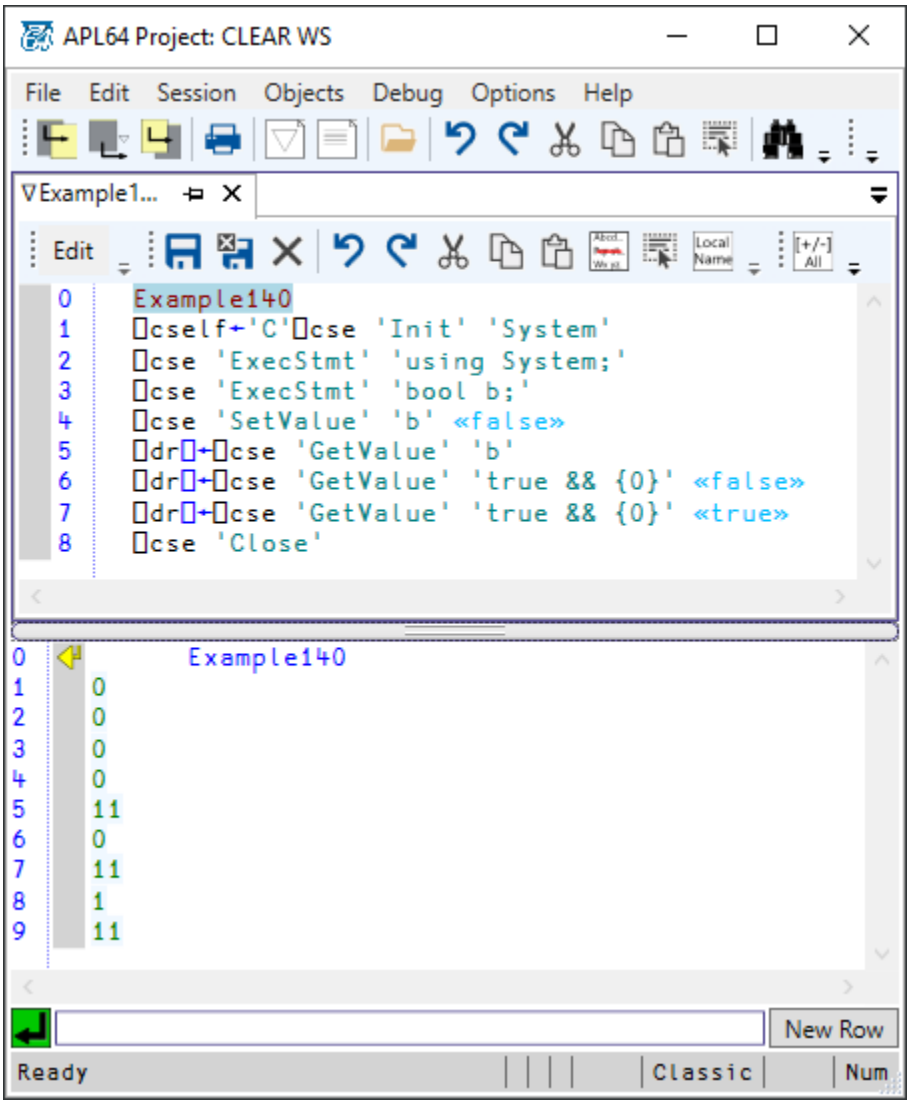


Example #140

Additional examples of sending bool values to a CSE instance.

```
Example140
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'bool b;'
⎕cse 'SetValue' 'b' «false»
⎕dr⎕←⎕cse 'GetValue' 'b'
```

```
⎕dr⎕←⎕cse 'GetValue' 'true && {0}' «false»
⎕dr⎕←⎕cse 'GetValue' 'true && {0}' «true»
⎕cse 'Close'
```



Example #117:

This example is analogous to example #116, except that a C# extension method is used to provide the coercion of an APL64 Boolean vector to a C# Boolean vector in C#. The '.ConvertVectorTo<T>()' C# extension method is defined in the top-level class of the CSE instance. These C# extension methods include a generalization which supports the programmer selection of the C# target type for the values of the C# vector object elements, in this case 'bool'.

The C# extension methods defined in this example include several convenient C# extension methods which can be helpful when exchanging information between APL64 and C#.

- public static T[] ToVector<T>(this T v)

- public static T[] ToVector<T>(this T[,] M, Int32 index, MatrixOrientation mo)
- public static T[,] ToMatrix<T>(this T[] V, MatrixOrientation mo)
- public static object[] ToObjectVector<T>(this T v)
- public static object[] ToObjectVector<T>(this T[] V)
- public static object[,] ToObjectMatrix<T>(this T[,] M)
- public static T[] ConvertVectorTo<T>(this Array ar)
- public static T[,] ConvertMatrixTo<T>(this Array ar)

These C# entension classes could also be defined in a .Net assembly created in Microsoft Visual Studio, compiled and loaded into the CSE using the LoadAssembly method and used for other CSE projects.

```
Example117;S
S←««
using System;
using System.Linq;
using System.ComponentModel;

public static bool[] ToBoolArraySimple(this Int32[] iV)
{
try
{
bool[] bV = new bool[iV.Length];
for (Int32 I = 0; I < iV.Length; I++)
{
bV[I] = Convert.ToBoolean(iV[I]);
}
return bV;
}
catch (Exception e)
{
throw new Exception("APLNext.CseExtensionMethods .ToBoolArray failed: " + e.Message);
}
}
public static bool[,] ToBoolArraySimple(this Int32[,] iA)
{
try
{
bool[,] bA = new bool[iA.GetLength(0),iA.GetLength(1)];
for (Int32 row = 0; row < iA.GetLength(0); row++)
{
for (Int32 col = 0; col < iA.GetLength(1); col++)
{
bA[row,col] = Convert.ToBoolean(iA[row,col]);
}
```

```
}
return bA;
}
catch (Exception e)
{
throw new Exception("APLNext.CseExtensionMethods .ToBoolArray failed: " + e.Message);
}
}
public static double[] ToDoubleArraySimple(this Int32[] iV)
{
try
{
double[] dV = new double[iV.Length];
for (Int32 I = 0; I < iV.Length; I++)
{
dV[I] = Convert.ToDouble(iV[I]);
}
return dV;
}
catch (Exception e)
{
throw new Exception("APLNext.CseExtensionMethods .ToDoubleArray failed: " + e.Message);
}
}
public static double[,] ToDoubleArraySimple(this Int32[,] iA)
{
try
{
double[,] dA = new double[iA.GetLength(0), iA.GetLength(1)];
for (Int32 row = 0; row < iA.GetLength(0); row++)
{
for (Int32 col = 0; col < iA.GetLength(1); col++)
{
dA[row, col] = Convert.ToDouble(iA[row, col]);
}
}
return dA;
}
catch (Exception e)
{
throw new Exception("APLNext.CseExtensionMethods .ToDoubleArray failed: " + e.Message);
}
}

public static T[] ToVector<T>(this T v)
```

```
{
try
{
T[] res = new T[] { v };
return res;
}
catch (Exception e)
{
throw new Exception("ToVector<T> failed: " + e.Message);
}
}
public static T[] ToVector<T>(this T[,] M, Int32 index, MatrixOrientation mo)
{
//index: index origin zero
try
{
Int32 L = M.GetLength((mo == MatrixOrientation.row) ? 1 : 0);
T[] res = new T[L];
for (Int32 I = 0; I < L; I++)
{
if (mo == MatrixOrientation.row)
res[I] = M[index, I];
else
res[I] = M[I, index];
}
return res;
}
catch (Exception e)
{
throw new Exception("ToVector<T> failed: " + e.Message);
}
}
public static T[,] ToMatrix<T>(this T[] V, MatrixOrientation mo)
{
try
{
T[,] res = new T[,] { };
if (mo == MatrixOrientation.row)
{
res = new T[1, V.Length];
for (Int32 col = 0; col < V.Length; col++)
{
res[0, col] = V[col];
}
}
```

```
else
{
res = new T[V.Length, 1];
for (Int32 row = 0; row < V.Length; row++)
{
res[row, 0] = V[row];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("ToMatrix<T> failed: " + e);
}
}
public static T[,] InsertRows<T>(this T[,] M, Int32 index, Int32 nRowsToInsert, bool insertBefore)
{
try
{
Int32 nRows = M.GetLength(0);
Int32 nCols = M.GetLength(1);
T[,] res = new T[nRows + nRowsToInsert, nCols];
Int32 rowOffset = index + (insertBefore ? -1 : 0);
for (Int32 row = 0; row < nRows; row++)
{
for (Int32 col = 0; col < nCols; col++)
{
res[row + ((row > rowOffset) ? nRowsToInsert : 0), col] = M[row, col];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("InsertRows failed: " + e.Message);
}
}
public static T[,] InsertCols<T>(this T[,] M, Int32 index, Int32 nColsToInsert, bool insertBefore)
{
try
{
Int32 nRows = M.GetLength(0);
Int32 nCols = M.GetLength(1);
T[,] res = new T[nRows, nCols + nColsToInsert];
Int32 colOffset = index + (insertBefore ? -1 : 0);
```

```
for (Int32 row = 0; row < nRows; row++)
{
for (Int32 col = 0; col < nCols; col++)
{
res[row, col + ((col > colOffset) ? nColsToInsert : 0)] = M[row, col];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("InsertCols failed: " + e.Message);
}
}

public static object[] ToObjectVector<T>(this T v)
{
try
{
object[] res = new object[] { v };
return res;
}
catch (Exception e)
{
throw new Exception("ToObject<T> failed: " + e.Message);
}
}
public static object[] ToObjectVector<T>(this T[] V)
{
try
{
object[] res = new object[V.Length];
for (Int32 I = 0; I < V.Length; I++)
{
res[I] = V[I];
}
return res;
}
catch (Exception e)
{
throw new Exception("ToObjectVector<T> failed: " + e.Message);
}
}
public static object[,] ToObjectMatrix<T>(this T[,] M)
{
```

```
try
{
Int32 nRows = M.GetLength(0);
Int32 nCols = M.GetLength(1);
object[,] res = new object[nRows, nCols];
for (Int32 row = 0; row < nRows; row++)
{
for (Int32 col = 0; col < nCols; col++)
{
res[row, col] = M[row, col];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("ToObjectMatrix<T> failed: " + e);
}
}

public static T[] Ravel<T>(this T[,] A)
{
try
{
Int32 nRows = A.GetLength(0);
Int32 nCols = A.GetLength(1);
T[] res = new T[A.Length];
for (Int32 row = 0; row < nRows; row++)
{
for (Int32 col = 0; col < nCols; col++)
{
res[(row * nRows) + col] = A[row, col];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("Ravel<T> failed: " + e.Message);
}
}
public static T[,] UpdateMatrix<T>(this T[,] A, T[] V, Int32 index, MatrixOrientation mo)
{
//index index origin zero
try
```

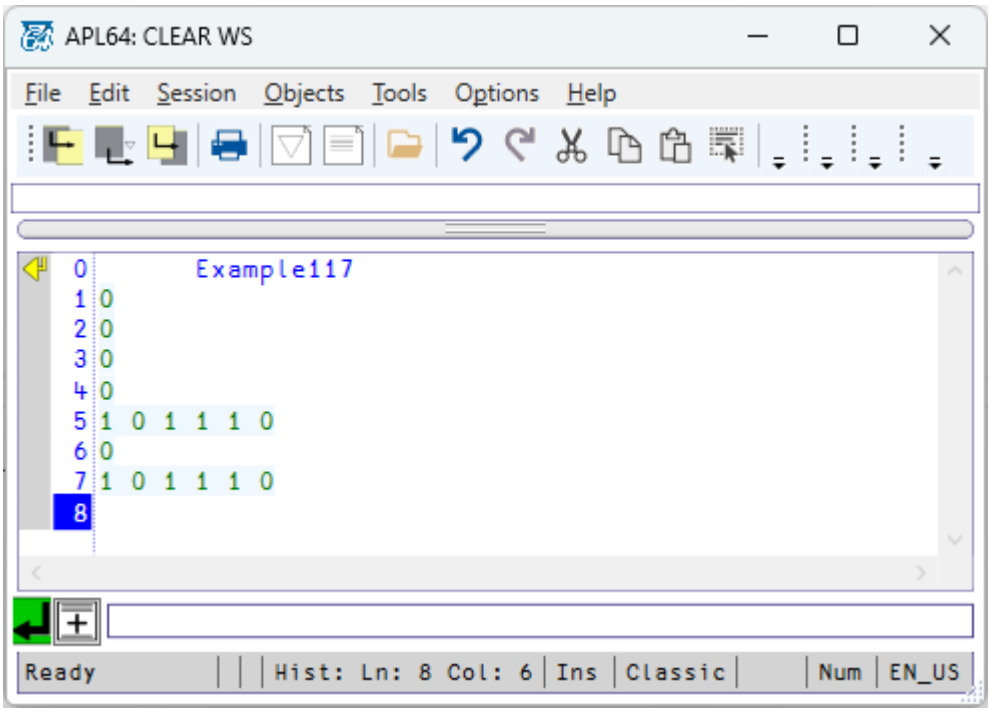
```
{
T[,] res = new T[A.GetLength(0), A.GetLength(1)];
Int32 nRows = A.GetLength(0);
Int32 nCols = A.GetLength(1);
for (Int32 row = 0; row < nRows; row++)
{
for (Int32 col = 0; col < nCols; col++)
{
if (mo == MatrixOrientation.row && row == index)
res[row, col] = V[col];
else if (mo == MatrixOrientation.col && col == index)
res[row, col] = V[row];
else
res[row, col] = A[row, col];
}
}
return res;
}
catch (Exception e)
{
throw new Exception("InsertVector<T> failed: " + e.Message);
}
}

public enum MatrixOrientation { row, col };
public enum IndexOrigin { zero, one };

public static Int32[] Iota(this Int32 nElts, IndexOrigin io)
{
try
{
Int32[] dV = new Int32[Math.Abs(nElts)];
dV = dV.Select((x, index) => index + ((io == IndexOrigin.zero) ? 0 : 1)).ToArray();
return dV;
}
catch (Exception e)
{
throw new Exception("Iota() failed: " + e.Message);
}
}

public static Int32[] VectorAdd(this Int32[] iV1, Int32[] iV2)
{
try
{
```

```
return iV1.Zip(iV2, (iv1, iv2) => iv1 + iv2).ToArray();
}
catch (Exception e)
{
throw new Exception("VectorAdd failed: " + e.Message);
}
}
public static double[] VectorAdd(this double[] iV1, double[] iV2)
{
try
{
return iV1.Zip(iV2, (iv1, iv2) => iv1 + iv2).ToArray();
}
catch (Exception e)
{
throw new Exception("VectorAdd failed: " + e.Message);
}
}
public static Int32[] VectorMult(this Int32[] iV1, Int32[] iV2)
{
try
{
return iV1.Zip(iV2, (iv1, iv2) => iv1 * iv2).ToArray();
}
catch (Exception e)
{
throw new Exception("VectorMult failed: " + e.Message);
}
}
public static double[] VectorMult(this double[] iV1, double[] iV2)
{
try
{
return iV1.Zip(iV2, (iv1, iv2) => iv1 * iv2).ToArray();
}
catch (Exception e)
{
throw new Exception("VectorMult failed: " + e.Message);
}
}
public static Int32[] ScalarAdd(this Int32[] iV, Int32 i)
{
if (0 == iV.Count())
return iV;
else
```

```
return (Int32[])iV.Select(k => k + i);
}
public static double[] ScalarAdd(this double[] iV, double i)
{
if (0 == iV.Count())
return new double[] { };
else
return (double[])iV.Select(k => k + i);
}
public static Int32[] ScalarMult(this Int32[] iV, Int32 i)
{
if (0 == iV.Count())
return iV;
else
return (Int32[])iV.Select(k => k * i);
}
public static double[] ScalarMult(this double[] iV, double i)
{
if (0 == iV.Count())
return iV;
else
return (double[])iV.Select(k => k * i);
}

public static T[] ConvertVectorTo<T>(this Array ar)
{
//http://www.extensionmethod.net/csharp/
try
{
T[] ret = new T[ar.Length];
TypeConverter tc = TypeDescriptor.GetConverter(typeof(T));
if (tc.CanConvertFrom(ar.GetValue(0).GetType()))
{
for (int i = 0; i < ar.Length; i++)
{
ret[i] = (T)tc.ConvertFrom(ar.GetValue(i));
}
}
else
{
tc = TypeDescriptor.GetConverter(ar.GetValue(0).GetType());
if (tc.CanConvertTo(typeof(T)))
{
for (int i = 0; i < ar.Length; i++)
{
```

```
ret[i] = (T)tc.ConvertTo(ar.GetValue(i), typeof(T));
}
}
else
throw new NotSupportedException();
}
return ret;
}
catch (Exception e)
{
throw new Exception("ConvertTo failed: " + e.Message);
}
}
public static T[,] ConvertMatrixTo<T>(this Array ar)
{
try
{
T[,] ret = new T[ar.GetLength(0), ar.GetLength(1)];
TypeConverter tc = TypeDescriptor.GetConverter(typeof(T));
if (tc.CanConvertFrom(ar.GetValue(0, 0).GetType()))
{
for (Int32 row = 0; row < ar.GetLength(0); row++)
{
for (Int32 col = 0; col < ar.GetLength(1); col++)
{
ret[row, col] = (T)tc.ConvertFrom(ar.GetValue(row, col));
}
}
}
else
{
tc = TypeDescriptor.GetConverter(ar.GetValue(0, 0).GetType());
if (tc.CanConvertTo(typeof(T)))
{
for (Int32 row = 0; row < ar.GetLength(0); row++)
{
for (Int32 col = 0; col < ar.GetLength(1); col++)
{
ret[row, col] = (T)tc.ConvertTo(ar.GetValue(row, col), typeof(T));
}
}
}
else
throw new NotSupportedException();
}
```

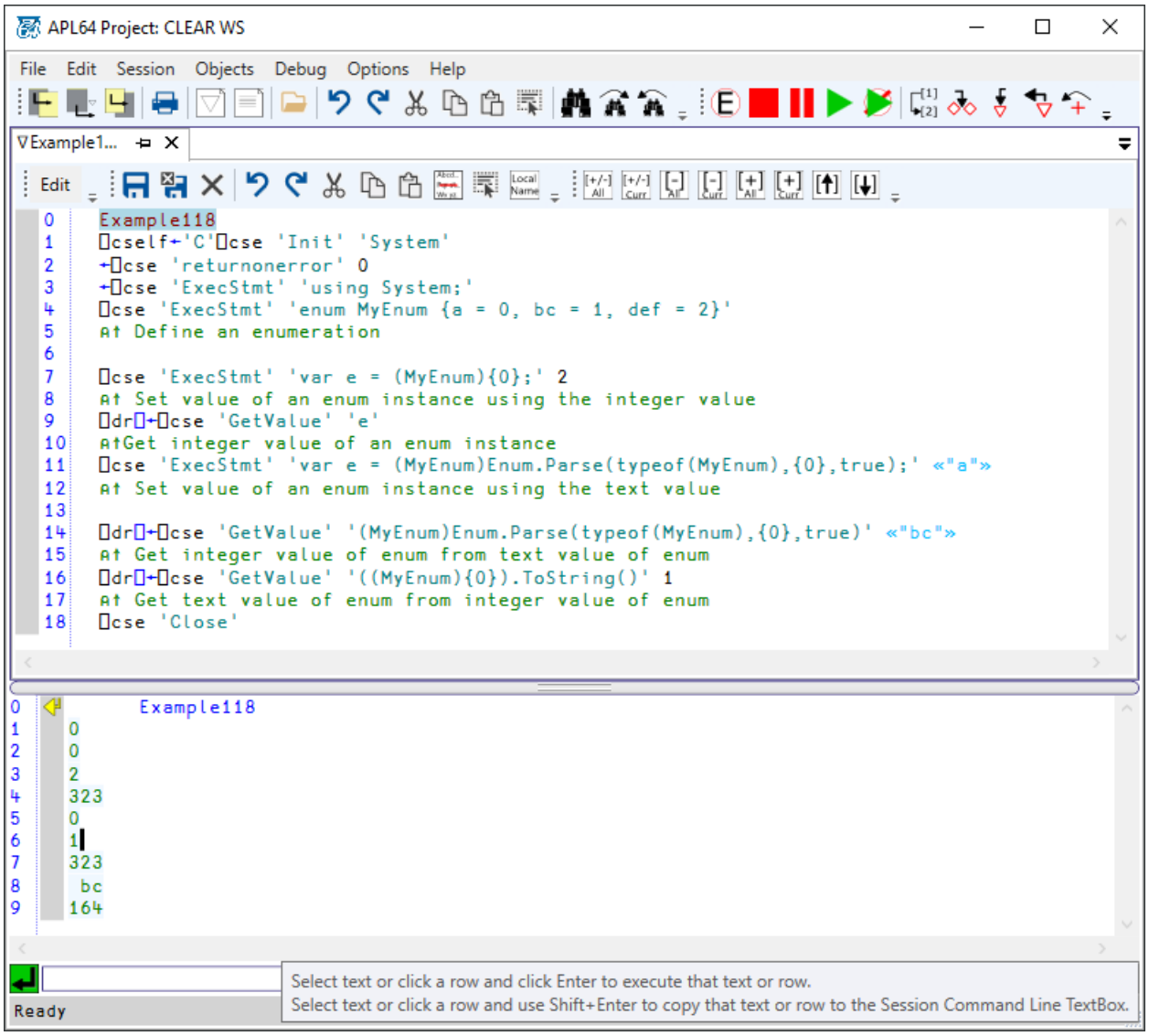
```
return ret;
}
catch (Exception e)
{
throw new Exception("ConvertTo failed: " + e.Message);
}
}
»»

⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'Exec' S

⎕cse 'ExecStmt' 'bool[] bV;' ⍝Define receiving C# variable
⎕cse 'ExecStmt' 'Int32[] iV;' ⍝Define intermediate C# receiveing variable
aplBV←1 0 1 1 1 0          ⍝Create APL+Win source variable
⎕cse 'SetValue' 'iV' aplBV    ⍝Set value of intermediate C# variable
⎕cse 'GetValue' 'iV'
⎕cse 'ExecStmt' 'bV = iV.ConvertVectorTo<bool>();' ⍝Use C# extension method
⍝↑ Coerce to appropriate array data type using C# extension method
⎕cse 'GetValue' 'bV'

⎕cse 'Close'
```



Example #118

This example illustrates creating and setting/getting the value of a .Net enumeration.

```
Example118
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'returnonerror' 0
←⎕cse 'ExecStmt' 'using System;'
⎕cse 'ExecStmt' 'enum MyEnum {a = 0, bc = 1, def = 2}'
⍝↑ Define an enumeration

⎕cse 'ExecStmt' 'var e = (MyEnum){0};' 2
⍝↑ Set value of an enum instance using the integer value
⎕dr⎕←⎕cse 'GetValue' 'e'
⍝↑Get integer value of an enum instance
⎕cse 'ExecStmt' 'var e = (MyEnum)Enum.Parse(typeof(MyEnum),{0},true);' «"a"»
⍝↑ Set value of an enum instance using the text value

⎕dr⎕←⎕cse 'GetValue' '(MyEnum)Enum.Parse(typeof(MyEnum),{0},true)' «"bc"»
⍝↑ Get integer value of enum from text value of enum
⎕dr⎕←⎕cse 'GetValue' '((MyEnum){0}).ToString()' 1
⍝↑ Get text value of enum from integer value of enum
⎕cse 'Close'
```

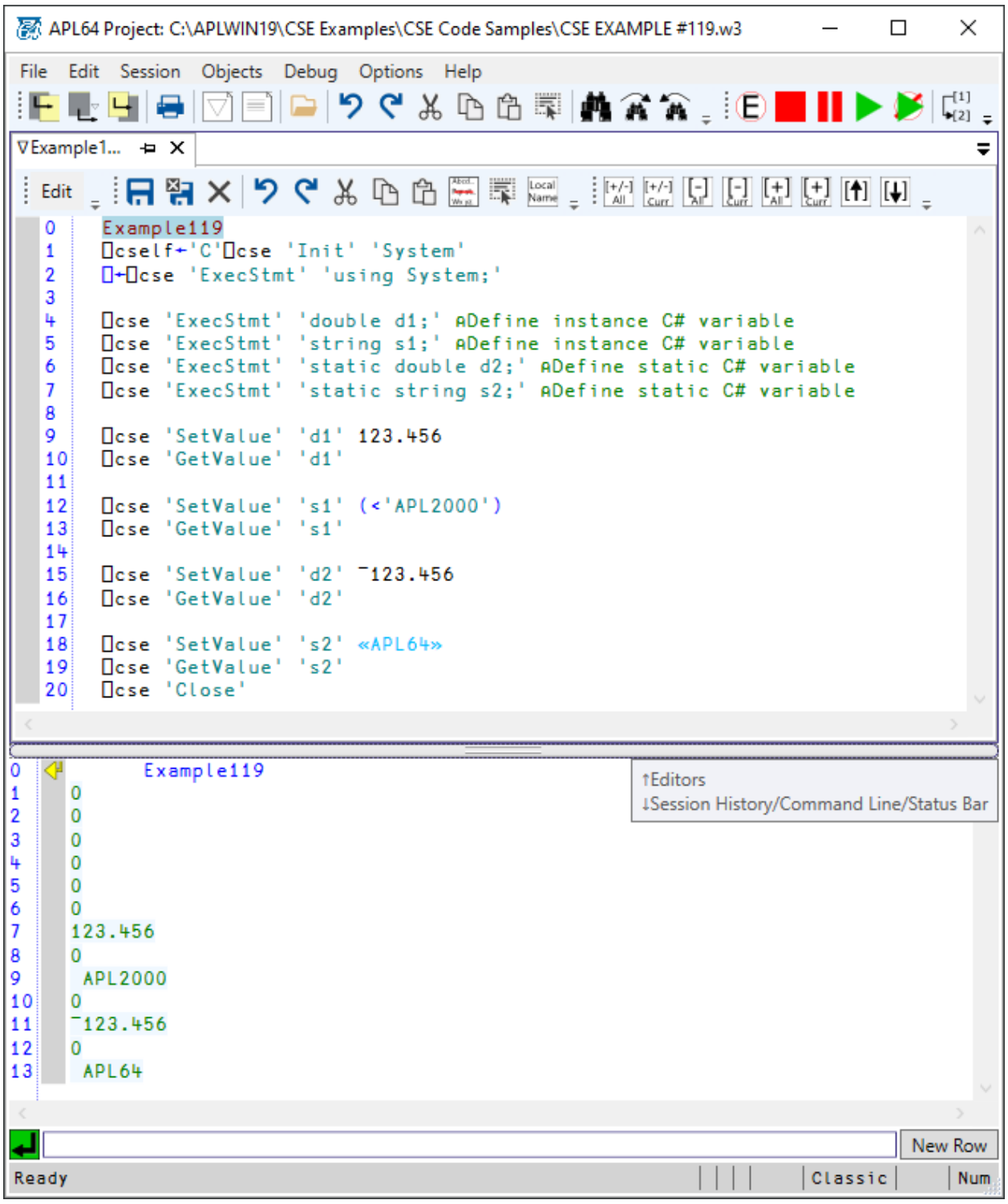


Example #119

In this example the values of instance and static variables in the scope of a CSE instance are set using the CSE ‘SetValue’ method.

```
Example119
⎕cself←'C'⎕cse 'Init' 'System'
⎕←⎕cse 'ExecStmt' 'using System;'

⎕cse 'ExecStmt' 'double d1;' ⍝Define instance C# variable
⎕cse 'ExecStmt' 'string s1;' ⍝Define instance C# variable
⎕cse 'ExecStmt' 'static double d2;' ⍝Define static C# variable
⎕cse 'ExecStmt' 'static string s2;' ⍝Define static C# variable

⎕cse 'SetValue' 'd1' 123.456
```

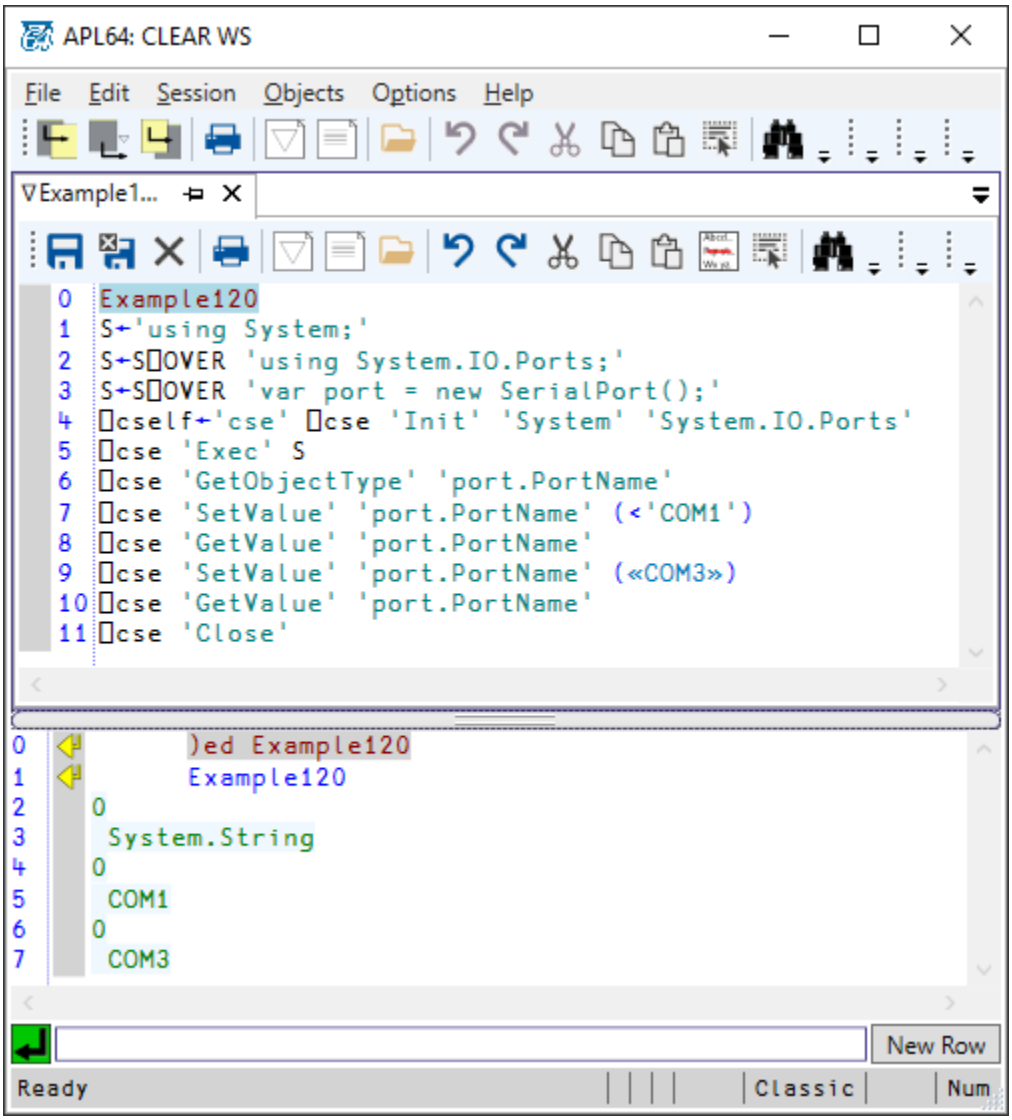
```
      cse 'GetValue' 'd1'

      cse 'SetValue' 's1' (<'APL2000')
      cse 'GetValue' 's1'

      cse 'SetValue' 'd2' ¯123.456
      cse 'GetValue' 'd2'

      cse 'SetValue' 's2' «APL64»
      cse 'GetValue' 's2'
      cse 'Close'
```



Example #120

In this example the CSE Button.Text property which has .Net string data type, is set by the CSE ‘SetValue’ method.

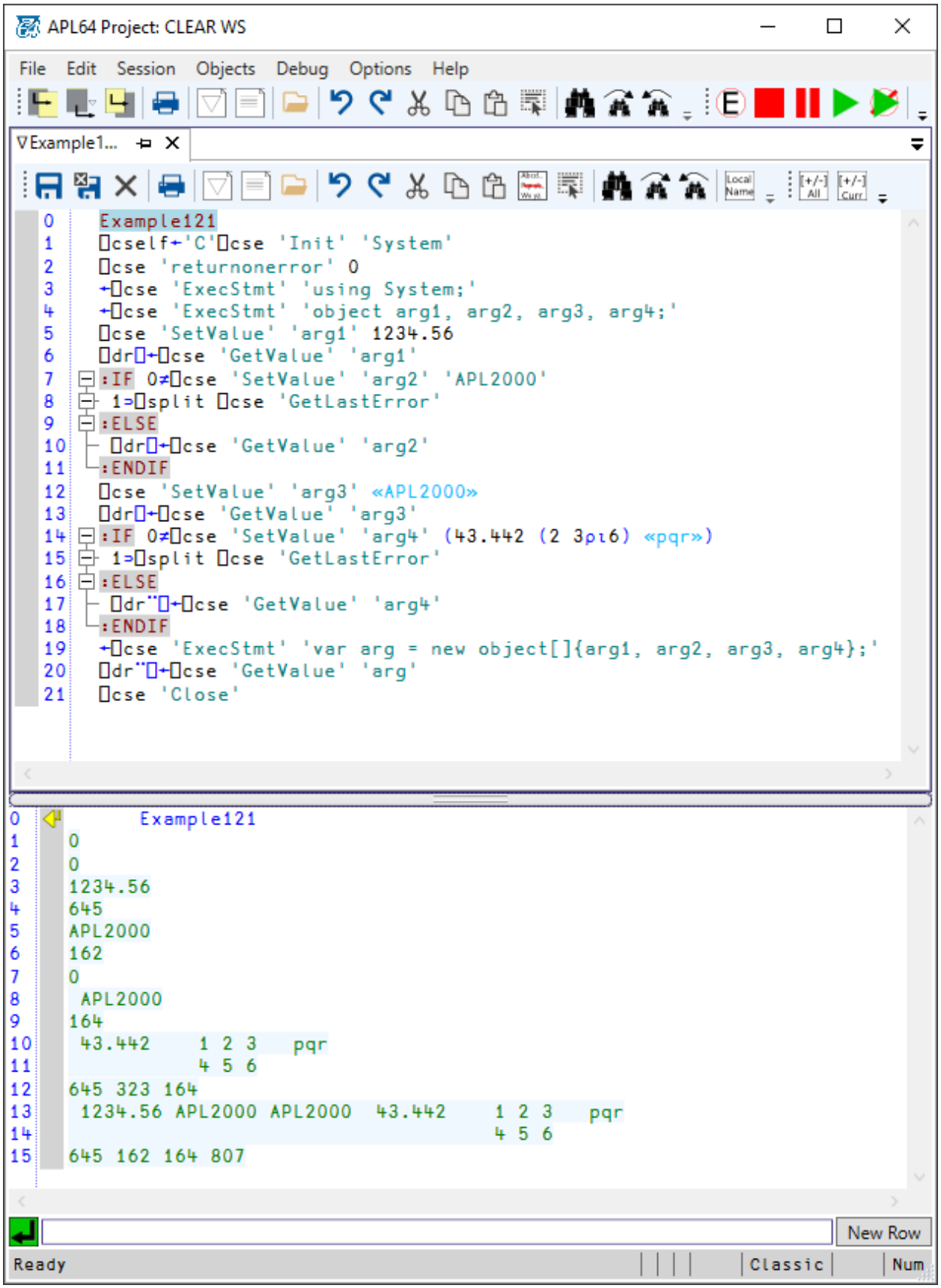
```
Example120
S←'using System;'
S←S□OVER 'using System.IO.Ports;'
S←S□OVER 'var port = new SerialPort();'
□cself←'cse' □cse 'Init' 'System' 'System.IO.Ports'
□cse 'Exec' S
□cse 'GetObjectType' 'port.PortName'
□cse 'SetValue' 'port.PortName' (<'COM1')
□cse 'GetValue' 'port.PortName'
□cse 'SetValue' 'port.PortName' («COM3»)
□cse 'GetValue' 'port.PortName'
□cse 'Close'
```



Example #121:

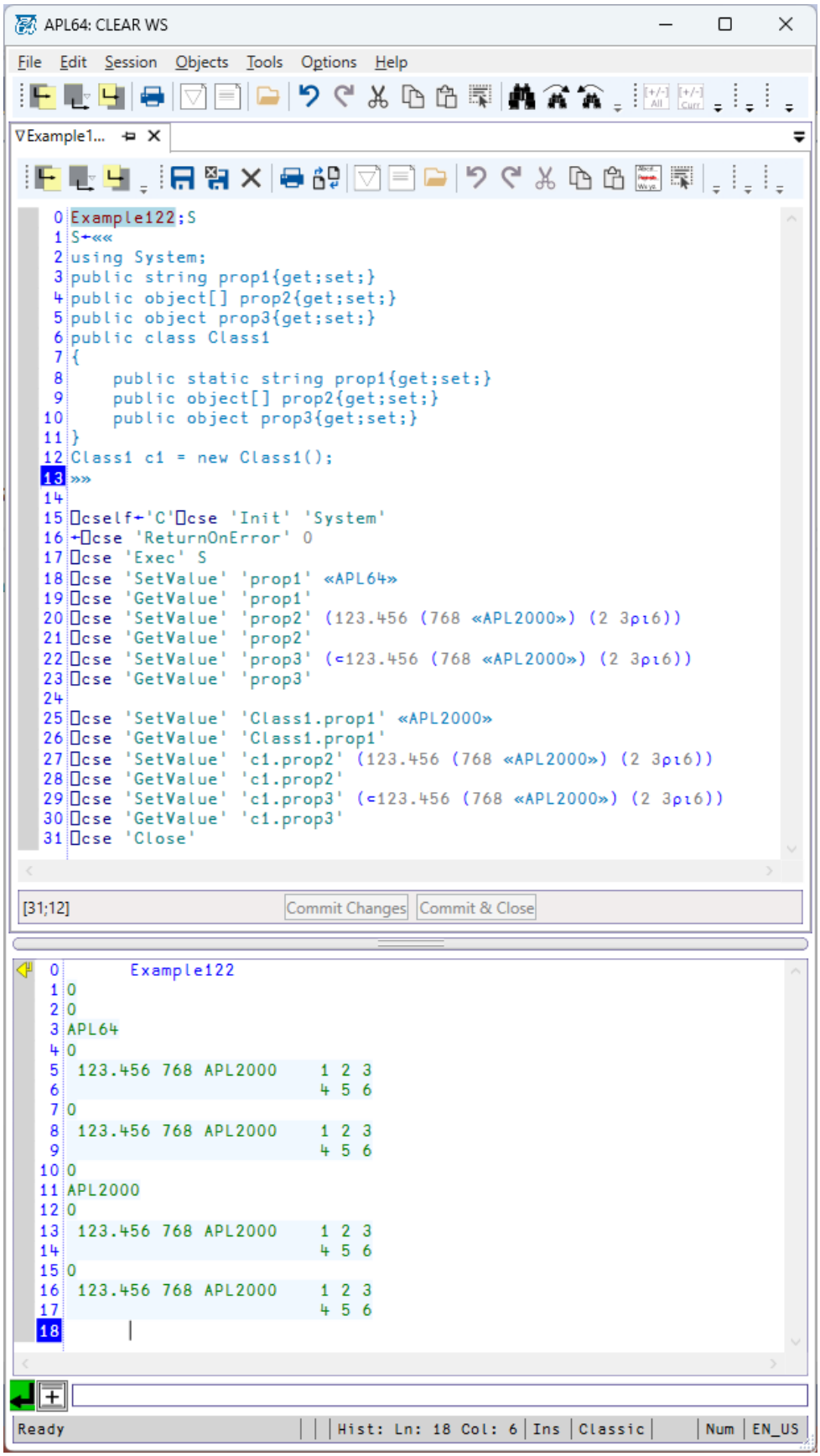
In this example several object type variables are created without initial values and their values are established using the CSE ‘SetValue’ method.

```
Example121
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'returnonerror' 0
←⎕cse 'ExecStmt' 'using System;'
←⎕cse 'ExecStmt' 'object arg1, arg2, arg3, arg4;'
⎕cse 'SetValue' 'arg1' 1234.56
⎕dr⎕←⎕cse 'GetValue' 'arg1'
:IF 0≠⎕cse 'SetValue' 'arg2' 'APL2000'
 1⊃⎕split ⎕cse 'GetLastError'
:ELSE
 ⎕dr⎕←⎕cse 'GetValue' 'arg2'
:ENDIF
⎕cse 'SetValue' 'arg3' «APL2000»
⎕dr⎕←⎕cse 'GetValue' 'arg3'
:IF 0≠⎕cse 'SetValue' 'arg4' (43.442 (2 3⍴⍳6) «pqr»)
 1⊃⎕split ⎕cse 'GetLastError'
:ELSE
 ⎕dr¨⎕←⎕cse 'GetValue' 'arg4'
:ENDIF
←⎕cse 'ExecStmt' 'var arg = new object[]{arg1, arg2, arg3, arg4};'
⎕dr¨⎕←⎕cse 'GetValue' 'arg'
⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
Example1...
0 Example121
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⎕cse 'returnonerror' 0
3 ←⎕cse 'ExecStmt' 'using System;'
4 ←⎕cse 'ExecStmt' 'object arg1, arg2, arg3, arg4;'
5 ⎕cse 'SetValue' 'arg1' 1234.56
6 ⎕dr⎕←⎕cse 'GetValue' 'arg1'
7 :IF 0≠⎕cse 'SetValue' 'arg2' 'APL2000'
8 1≥⎕split ⎕cse 'GetLastError'
9 :ELSE
10 ⎕dr⎕←⎕cse 'GetValue' 'arg2'
11 :ENDIF
12 ⎕cse 'SetValue' 'arg3' «APL2000»
13 ⎕dr⎕←⎕cse 'GetValue' 'arg3'
14 :IF 0≠⎕cse 'SetValue' 'arg4' (43.442 (2 3⍴⍳6) «pqr»)
15 1≥⎕split ⎕cse 'GetLastError'
16 :ELSE
17 ⎕dr¨⎕←⎕cse 'GetValue' 'arg4'
18 :ENDIF
19 ←⎕cse 'ExecStmt' 'var arg = new object[]{arg1, arg2, arg3, arg4};'
20 ⎕dr¨⎕←⎕cse 'GetValue' 'arg'
21 ⎕cse 'Close'
0 Example121
1 0
2 0
3 1234.56
4 645
5 APL2000
6 162
7 0
8 APL2000
9 164
10 43.442 1 2 3 pqr
11 4 5 6
12 645 323 164
13 1234.56 APL2000 APL2000 43.442 1 2 3 pqr
14 4 5 6
15 645 162 164 807
New Row
Ready
Classic
Num

Example #122:
In this example C# instance and static property values are set from APL64 using the CSE 'SetValue' method. A C# static property defined in the scope of the CSE instance is not illustrated, and should not be used because there is no available class name (of the CSE instance) with which to reference such a property. C# instance and static properties of well-defined C# classes are fully supported by the CSE.

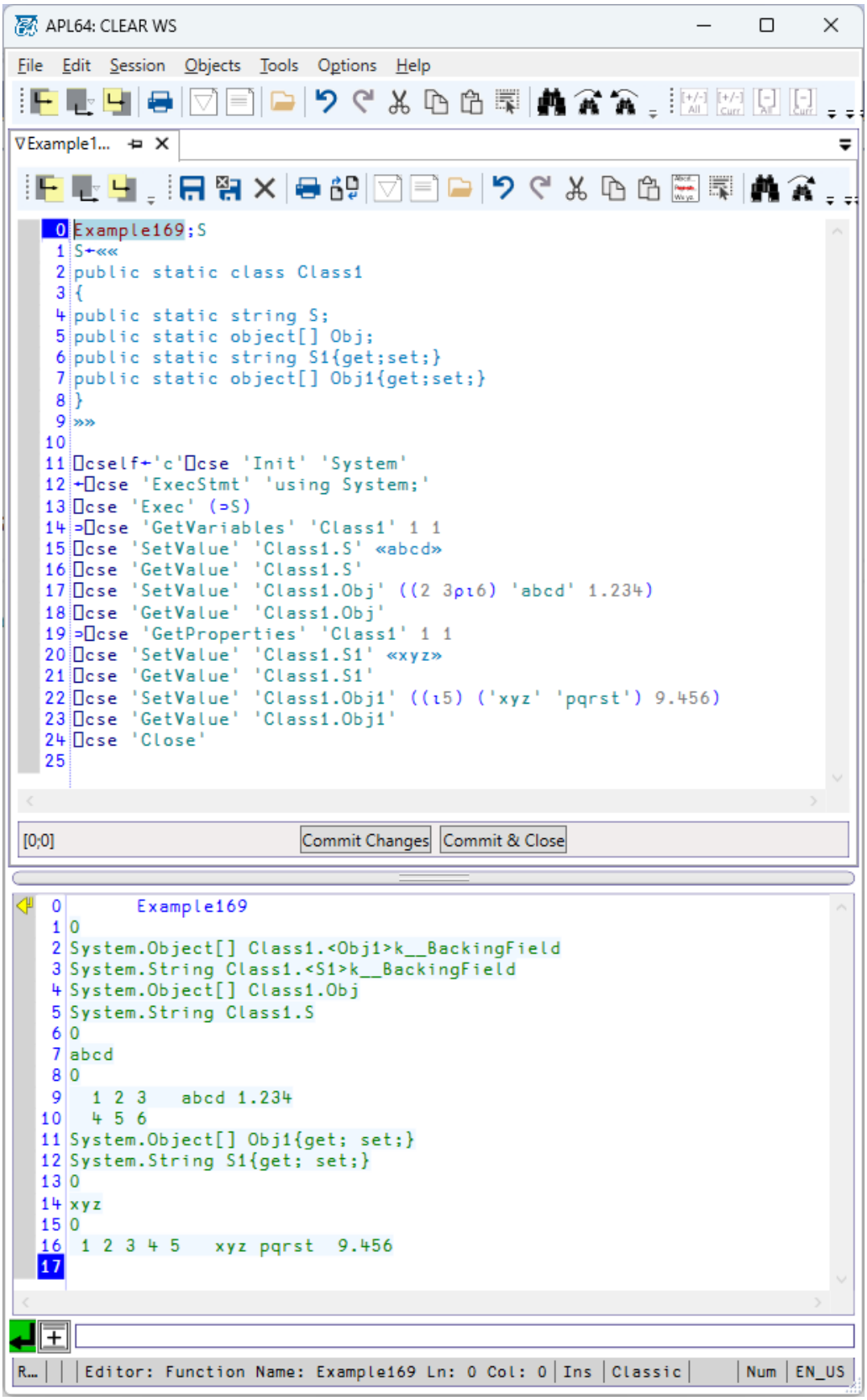
```
Example122;S
S←««
using System;
public string prop1{get;set;}
public object[] prop2{get;set;}
public object prop3{get;set;}
public class Class1
{
   public static string prop1{get;set;}
   public object[] prop2{get;set;}
   public object prop3{get;set;}
}
Class1 c1 = new Class1();
»»

□cself←'C'□cse 'Init' 'System'
←□cse 'ReturnOnError' 0
□cse 'Exec' S
□cse 'SetValue' 'prop1' «APL64»
□cse 'GetValue' 'prop1'
□cse 'SetValue' 'prop2' (123.456 (768 «APL2000») (2 3⍴⍳6))
□cse 'GetValue' 'prop2'
□cse 'SetValue' 'prop3' (⊂123.456 (768 «APL2000») (2 3⍴⍳6))
□cse 'GetValue' 'prop3'

□cse 'SetValue' 'Class1.prop1' «APL2000»
□cse 'GetValue' 'Class1.prop1'
□cse 'SetValue' 'c1.prop2' (123.456 (768 «APL2000») (2 3⍴⍳6))
□cse 'GetValue' 'c1.prop2'
□cse 'SetValue' 'c1.prop3' (⊂123.456 (768 «APL2000») (2 3⍴⍳6))
□cse 'GetValue' 'c1.prop3'
□cse 'Close'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
∇Example1...
0 Example122;S
1 S←««
2 using System;
3 public string prop1{get;set;}
4 public object[] prop2{get;set;}
5 public object prop3{get;set;}
6 public class Class1
7 {
8 public static string prop1{get;set;}
9 public object[] prop2{get;set;}
10 public object prop3{get;set;}
11 }
12 Class1 c1 = new Class1();
13 »»
14
15 ⎕cself←'C'⎕cse 'Init' 'System'
16 ←⎕cse 'ReturnOnError' 0
17 ⎕cse 'Exec' S
18 ⎕cse 'SetValue' 'prop1' «APL64»
19 ⎕cse 'GetValue' 'prop1'
20 ⎕cse 'SetValue' 'prop2' (123.456 (768 «APL2000») (2 3⍴⍳6))
21 ⎕cse 'GetValue' 'prop2'
22 ⎕cse 'SetValue' 'prop3' (⊂123.456 (768 «APL2000») (2 3⍴⍳6))
23 ⎕cse 'GetValue' 'prop3'
24
25 ⎕cse 'SetValue' 'Class1.prop1' «APL2000»
26 ⎕cse 'GetValue' 'Class1.prop1'
27 ⎕cse 'SetValue' 'c1.prop2' (123.456 (768 «APL2000») (2 3⍴⍳6))
28 ⎕cse 'GetValue' 'c1.prop2'
29 ⎕cse 'SetValue' 'c1.prop3' (⊂123.456 (768 «APL2000») (2 3⍴⍳6))
30 ⎕cse 'GetValue' 'c1.prop3'
31 ⎕cse 'Close'
[31;12]
Commit Changes
Commit & Close
0 Example122
1 0
2 0
3 APL64
4 0
5 123.456 768 APL2000 1 2 3
6 4 5 6
7 0
8 123.456 768 APL2000 1 2 3
9 4 5 6
10 0
11 APL2000
12 0
13 123.456 768 APL2000 1 2 3
14 4 5 6
15 0
16 123.456 768 APL2000 1 2 3
17 4 5 6
18
Ready
Hist: Ln: 18 Col: 6
Ins
Classic
Num
EN_US

Example #169:
This example illustrates a script which creates a C# static class with static variables (fields) and properties. The CSE ‘SetValue’ method is used to set the value of these variables and properties from APL64 values.

The result of the CSE ‘GetVariables’ method includes the two static variables defined in the Class1 static class. The result of the CSE ‘GetVariables’ method also includes the two implicit ‘backing’ fields for the two static properties which were defined in the class using the implicit C# notation ‘{get; set;}’.

```
Example169;S
S←««
public static class Class1
{
public static string S;
public static object[] Obj;
public static string S1{get;set;}
public static object[] Obj1{get;set;}
}
»»

□cself←'c'□cse 'Init' 'System'
←□cse 'ExecStmt' 'using System;'
□cse 'Exec' (⊃S)
⊃□cse 'GetVariables' 'Class1' 1 1
□cse 'SetValue' 'Class1.S' «abcd»
□cse 'GetValue' 'Class1.S'
□cse 'SetValue' 'Class1.Obj' ((2 3⍴⍳6) 'abcd' 1.234)
□cse 'GetValue' 'Class1.Obj'
⊃□cse 'GetProperties' 'Class1' 1 1
□cse 'SetValue' 'Class1.S1' «xyz»
□cse 'GetValue' 'Class1.S1'
□cse 'SetValue' 'Class1.Obj1' ((⍳5) ('xyz' 'pqrst') 9.456)
□cse 'GetValue' 'Class1.Obj1'
□cse 'Close'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Example1...
0 Example169;S
1 S←««
2 public static class Class1
3 {
4 public static string S;
5 public static object[] Obj;
6 public static string S1{get;set;}
7 public static object[] Obj1{get;set;}
8 }
9 »»
10
11 ⎕cself←'c'⎕cse 'Init' 'System'
12 ←⎕cse 'ExecStmt' 'using System;'
13 ⎕cse 'Exec' (⊃S)
14 ⊃⎕cse 'GetVariables' 'Class1' 1 1
15 ⎕cse 'SetValue' 'Class1.S' «abcd»
16 ⎕cse 'GetValue' 'Class1.S'
17 ⎕cse 'SetValue' 'Class1.Obj' ((2 3⍴⍳6) 'abcd' 1.234)
18 ⎕cse 'GetValue' 'Class1.Obj'
19 ⊃⎕cse 'GetProperties' 'Class1' 1 1
20 ⎕cse 'SetValue' 'Class1.S1' «xyz»
21 ⎕cse 'GetValue' 'Class1.S1'
22 ⎕cse 'SetValue' 'Class1.Obj1' ((⍳5) ('xyz' 'pqrst') 9.456)
23 ⎕cse 'GetValue' 'Class1.Obj1'
24 ⎕cse 'Close'
25
[0;0]
Commit Changes
Commit & Close
0 Example169
1 0
2 System.Object[] Class1.<Obj1>k__BackingField
3 System.String Class1.<S1>k__BackingField
4 System.Object[] Class1.Obj
5 System.String Class1.S
6 0
7 abcd
8 0
9 1 2 3 abcd 1.234
10 4 5 6
11 System.Object[] Obj1{get; set;}
12 System.String S1{get; set;}
13 0
14 xyz
15 0
16 1 2 3 4 5 xyz pqrst 9.456
17
R...
Editor: Function Name: Example169 Ln: 0 Col: 0
Ins
Classic
Num
EN_US

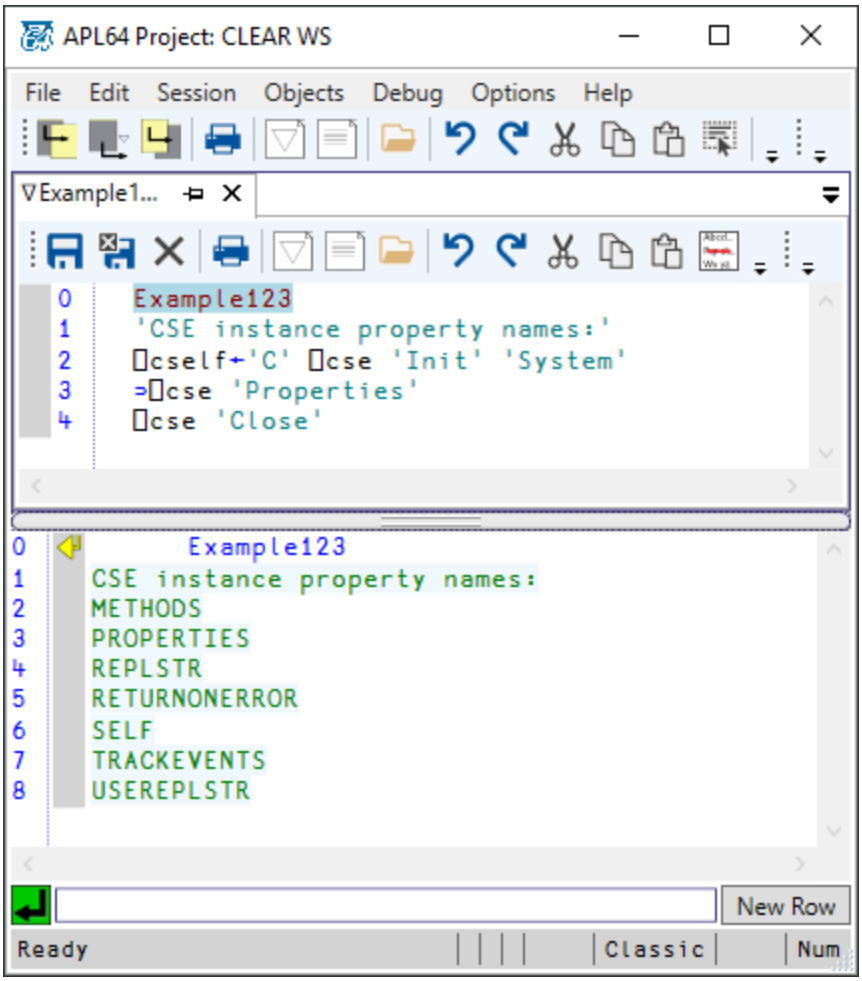
CSE Properties Applicable to a CSE Instance

The CSE 'properties' property returns a vector of text vectors containing the names of the properties supported by an instance of the CSE. The action of getting or setting the value of a CSE property is always applied to a specified instance of the CSE object. This specified CSE object instance may be explicitly named in the left argument to the □cse system function. The left argument to the □cse system function is optional and assumed to be the current value e CSE □cself system variable.

Obtaining the value or setting the value of a CSE property should be performed after the CSE 'Init' method is executed and after the 'using System;' C# statement is executed using the CSE 'Exec', 'ExecFile' or 'ExecStmt' methods. This is necessary so that the CSE can perform the proper coercion of values between APL64 and C#.

Example #123

```
Example123
'CSE instance property names:'
□cself←'C' □cse 'Init' 'System'
⊃□cse 'Properties'
□cse 'Close'
```

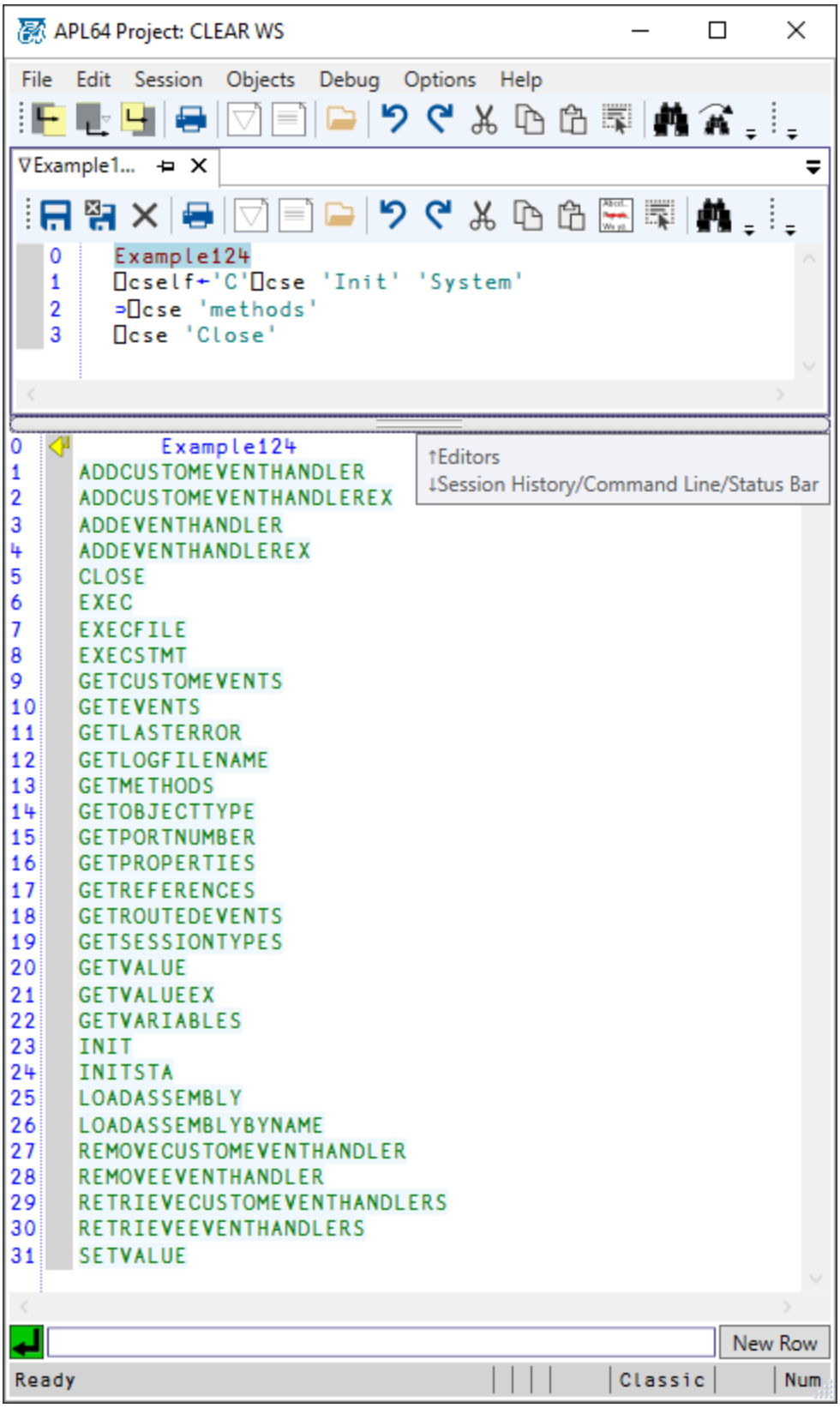


CSE methods Property

The CSE 'methods' property is a read-only property which returns a vector of text vectors containing the names of the methods supported by the CSE. Example #124:

Example124

```
Example124
⎕cself←'C'⎕cse 'Init' 'System'
⊃⎕cse 'methods'
⎕cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
∇Example1...
0 Example124
1 ⎕cself←'C'⎕cse 'Init' 'System'
2 ⍎⎕cse 'methods'
3 ⎕cse 'Close'
0 Example124
1 ADDCUSTOMEVENTHANDLER
2 ADDCUSTOMEVENTHANDLEREX
3 ADDEVENTHANDLER
4 ADDEVENTHANDLEREX
5 CLOSE
6 EXEC
7 EXECFILE
8 EXECSTMT
9 GETCUSTOMEVENTS
10 GETEVENTS
11 GETLASTERROR
12 GETLOGFILENAME
13 GETMETHODS
14 GETOBJECTTYPE
15 GETPORTNUMBER
16 GETPROPERTIES
17 GETREFERENCES
18 GETROUTEDEVENTS
19 GETSESSIONTYPES
20 GETVALUE
21 GETVALUEEX
22 GETVARIABLES
23 INIT
24 INITSTA
25 LOADASSEMBLY
26 LOADASSEMBLYBYNAME
27 REMOVECUSTOMEVENTHANDLER
28 REMOVEEVENTHANDLER
29 RETRIEVECUSTOMEVENTHANDLERS
30 RETRIEVEEVENTHANDLERS
31 SETVALUE
↑Editors
↓Session History/Command Line/Status Bar
New Row
Ready
Classic
Num

CSE replstr property

The 'replstr' property applies only to the CSE 'ExecStmt' and 'GetValue' methods when the value substitution option is used and when the value of the CSE 'usereplstr' property is 1.

The shape of the value of this property is a two-column array with one row for each replacement that should occur. Column #1 of this array contains the characters or character vectors to be replaced and column #2 contains the corresponding replacement characters or character vectors.

When converting between APL64 and .Net, the ReplStr property value will be applied to a string values (scalar or array) and character values (scalar or rank 1 array).

If the value of the CSE 'usereplstr' is 1, the character vectors in the substitution values for the 'ExecStmt' or 'GetValue' method which occur in the 1st column of the 'replstrs' property value will be replaced by the character vectors in the corresponding 2nd column of the 'replstr' property value.

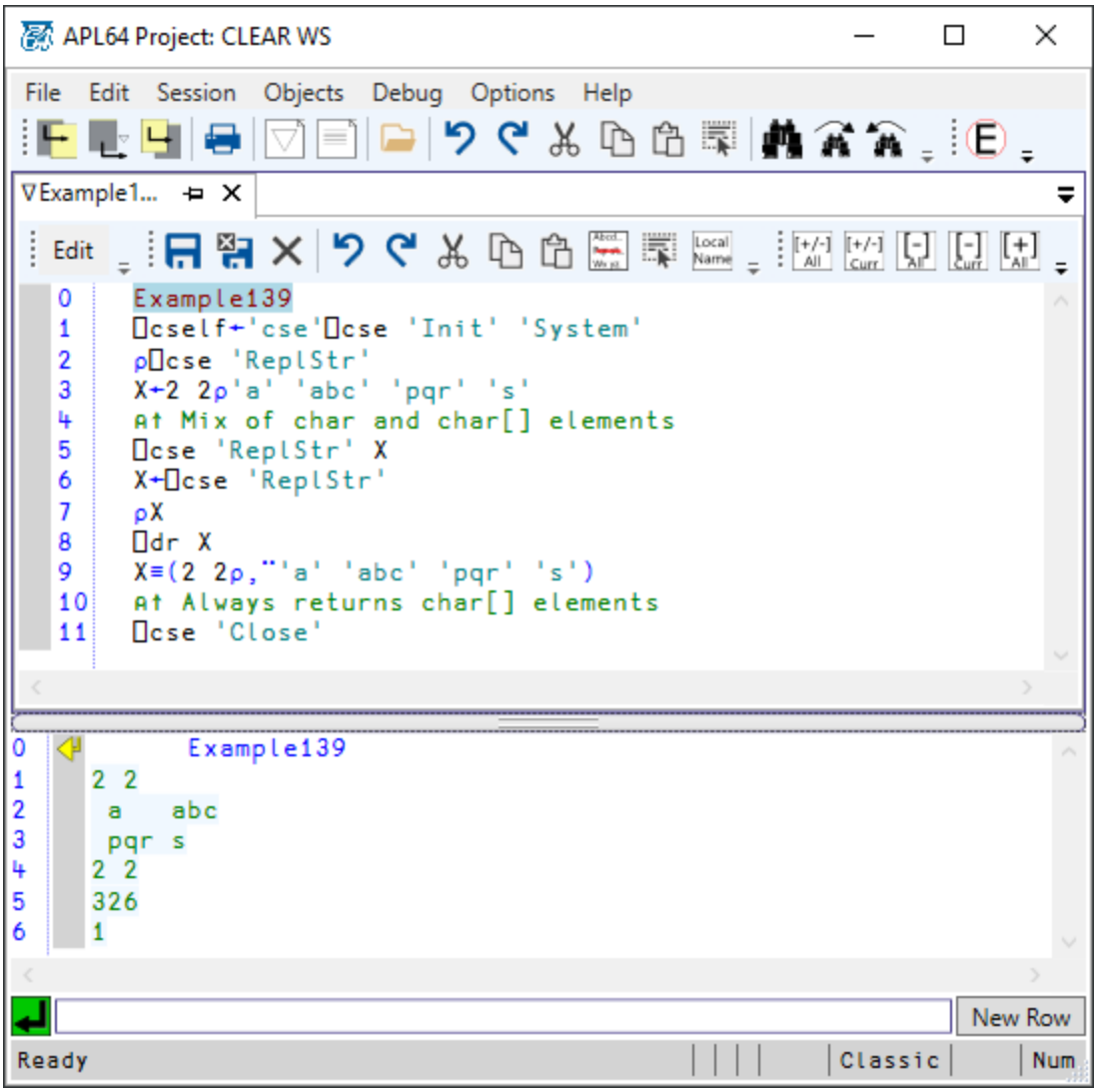
The value of the ReplStr property is not saved in the APL64 xml-format configuration file. This value must be explicitly set by the APL64 programmer before use as the ReplStr property. The default value of the ReplStr property is 0 2ρ' '.

The value of the elements of the ReplStr property value returned by executing ⎕cse 'ReplStr' will always be character vectors.

Example #139

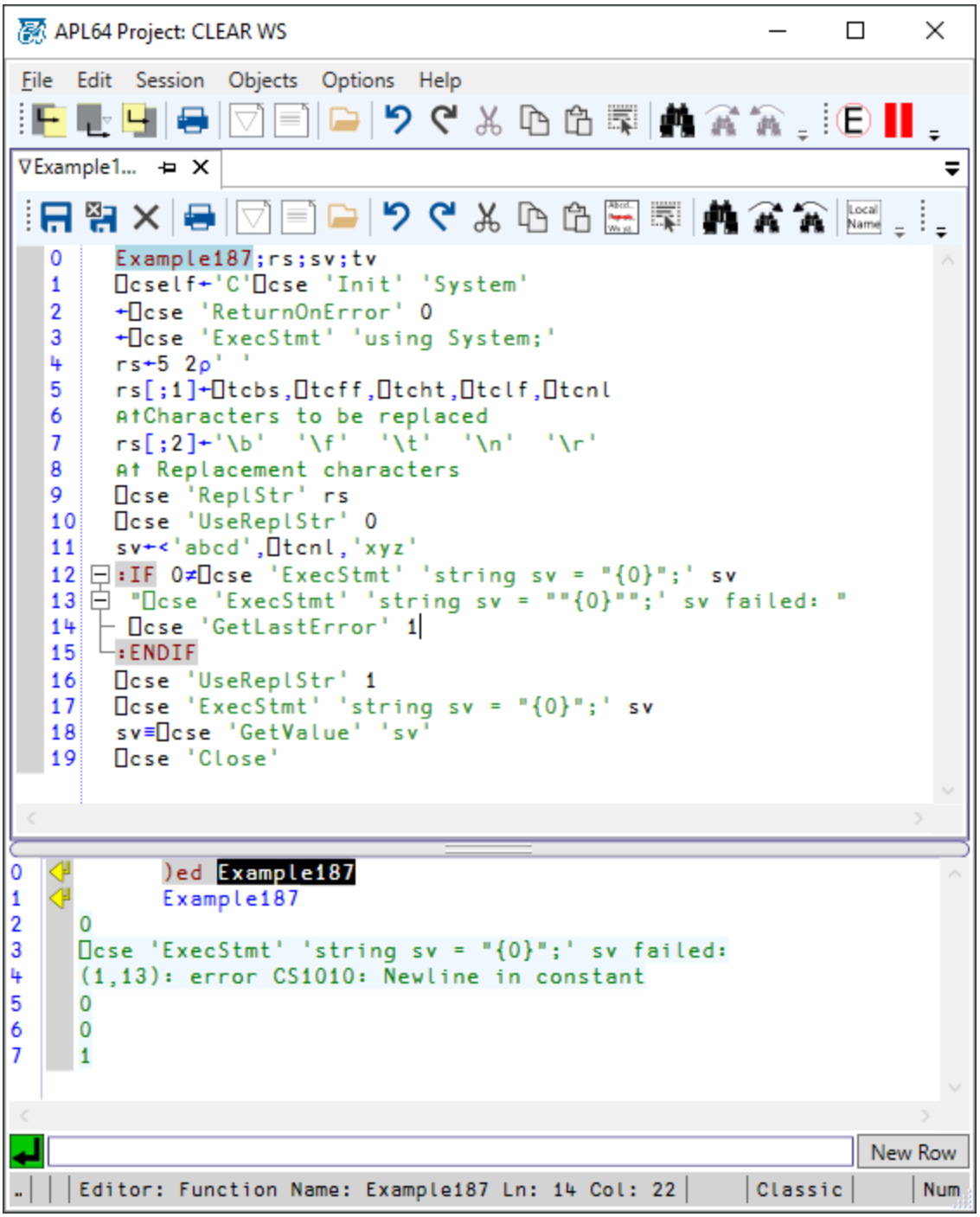
This example illustrates setting and getting the value of the ReplStr property.

```
Example139
⎕cself←'cse'⎕cse 'Init' 'System'
ρ⎕cse 'ReplStr'
X←2 2ρ'a' 'abc' 'pqr' 's'
⍥⍞↑ Mix of char and char[] elements
⎕cse 'ReplStr' X
X←⎕cse 'ReplStr'
ρX
⎕dr X
X≡(2 2ρ,¨'a' 'abc' 'pqr' 's')
⍥⍞↑ Always returns char[] elements
⎕cse 'Close'
```



Example #187: In this example the CSE APL64 value substitution feature is used to set a C# variable value containing a newline character from the APL64 session. When this action is attempted without using the CSE 'usereplstr' and 'replstr' properties, a .Net exception occurs. Using the CSE 'usereplstr' and 'replstr' properties, the ⎕tcnl character in the APL64 variable is replaced with '\r' so that the newline character is 'escaped' in the value of the target C# variable.

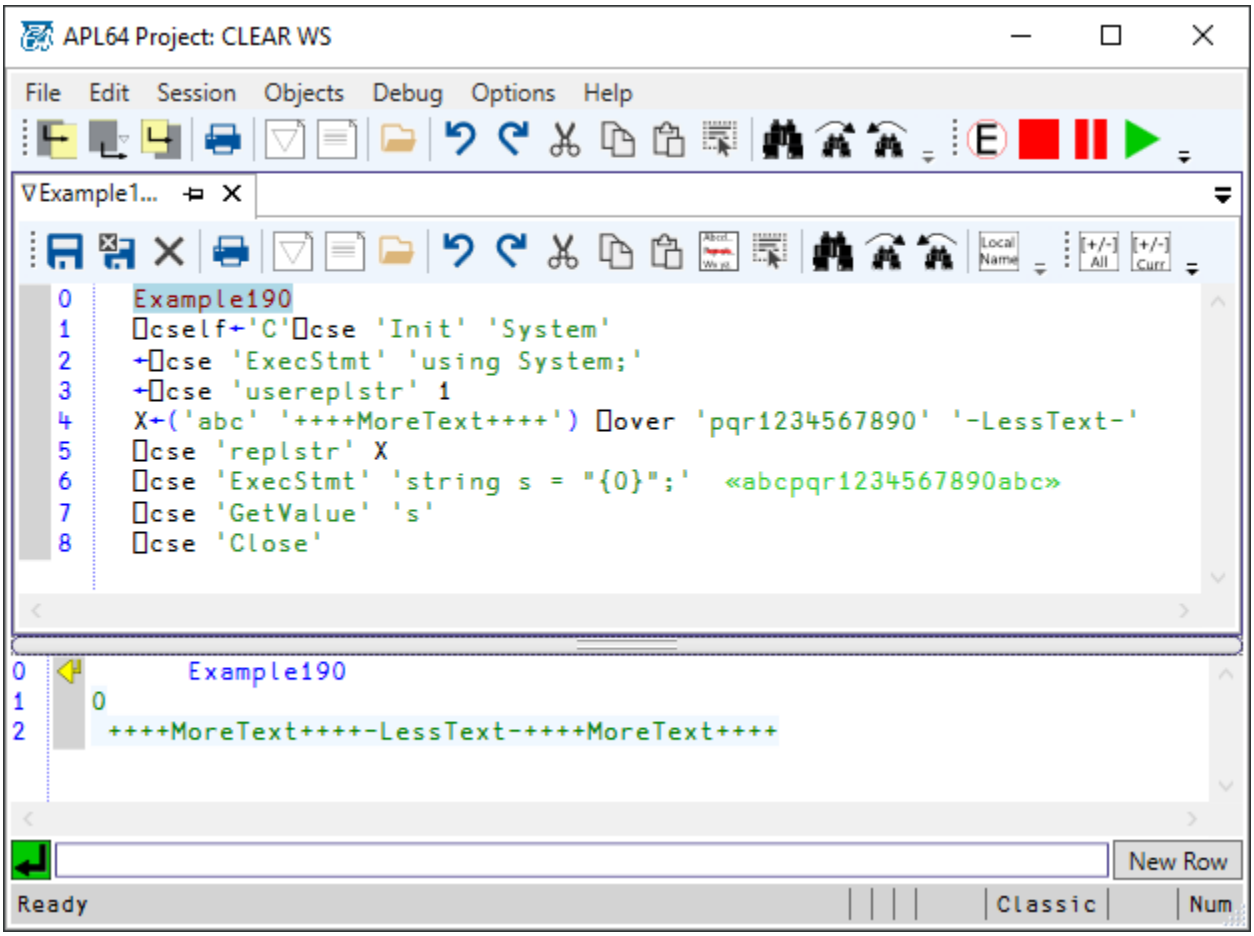
```
Example187;rs;sv;tv
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ReturnOnError' 0
←⎕cse 'ExecStmt' 'using System;'
rs←5 2⍴' '
rs[;1]←⎕tcbs,⎕tcff,⎕tcht,⎕tclf,⎕tcnl
⍝↑Characters to be replaced
rs[;2]←'\b'  '\f'  '\t'  '\n'  '\r'
⍝↑ Replacement characters
⎕cse 'ReplStr' rs
⎕cse 'UseReplStr' 0
sv←<'abcd',⎕tcnl,'xyz'
:IF 0≠⎕cse 'ExecStmt' 'string sv = "{0}";' sv
 "⎕cse 'ExecStmt' 'string sv = ""{0}"";' sv failed: "
 ⎕cse 'GetLastError' 1
:ENDIF
⎕cse 'UseReplStr' 1
⎕cse 'ExecStmt' 'string sv = "{0}";' sv
sv≡⎕cse 'GetValue' 'sv'
⎕cse 'Close'
```



Example #190: The CSE 'usereplstr' and 'replstr' properties can also be used replace more or less text.

```
Example190
⎕cself←'C'⎕cse 'Init' 'System'
←⎕cse 'ExecStmt' 'using System;'
←⎕cse 'usereplstr' 1
X←('abc' '++++MoreText++++') ⎕over 'pqr1234567890' '-LessText-'
⎕cse 'replstr' X
```

```
⎕cse 'ExecStmt' 'string s = "{0}";'  «abcpqr1234567890abc»
⎕cse 'GetValue' 's'
⎕cse 'Close'
```



CSE returnonerror property

The CSE 'returnonerror' property affects the result of these CSE methods:

- AddCustomEventHandler
- AddCustomEventHandlerEx
- AddEventHandler
- AddEventHandlerEx
- Exec
- ExecFile
- ExecStmt
- RemoveCustomEventHandler
- RemoveEventHandler
- SetValue

The CSE 'returnonerror' property does not affect the result of the CSE 'GetValue' method since the purpose of this method is to return a data value which could not be distinguished from an error code.

The purpose of the CSE 'returnonerror' property is to provide support for two methods of exception handling in APL64. The CSE 'GetLastError' method will provide the most recent C# exception description for both values of the CSE 'returnonerror' property.

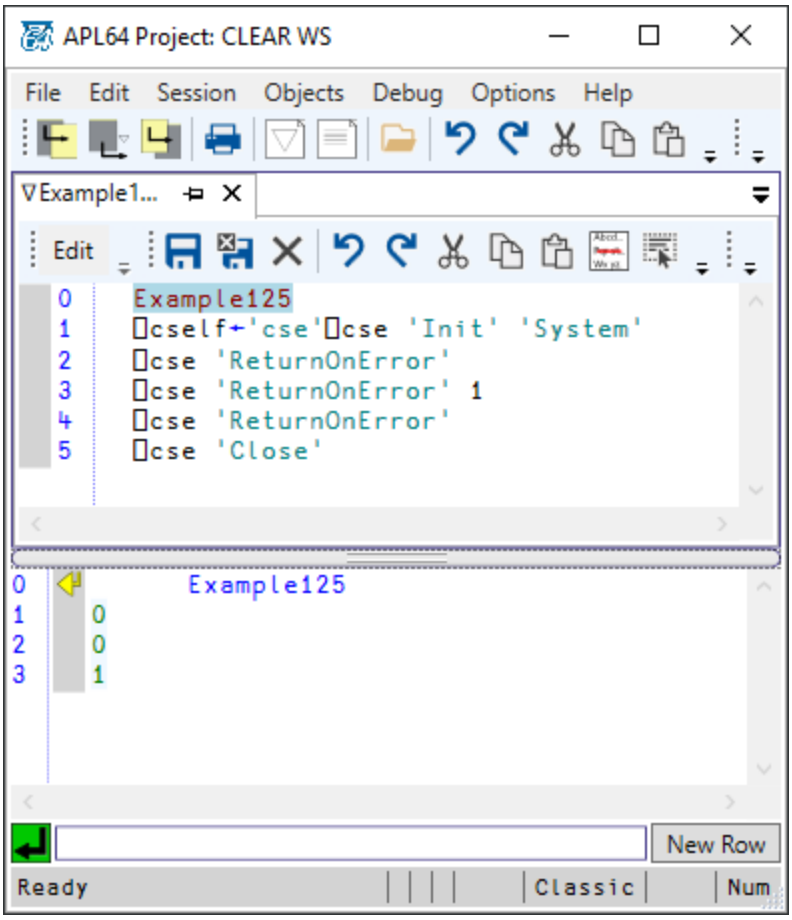
Note that most of the examples in this document use the default value of the CSE 'returnonerror' property.

	Result of the affected CSE methods	
returnonerror	No C# Exception Occurred	C# Exception Occurred
0 (Default)	0	Explicit result -1. No APL64 exception will be thrown so that C# exceptions can be distinguished from APL64 exceptions. The APL64 □DM value will not be affected.
1	0	No explicit result. An APL64 exception will be thrown and the APL64 □DM value will contain the C# exception description.

Example #125:

When setting the value of the CSE 'returnonerror' property the prior value of this property will be returned.

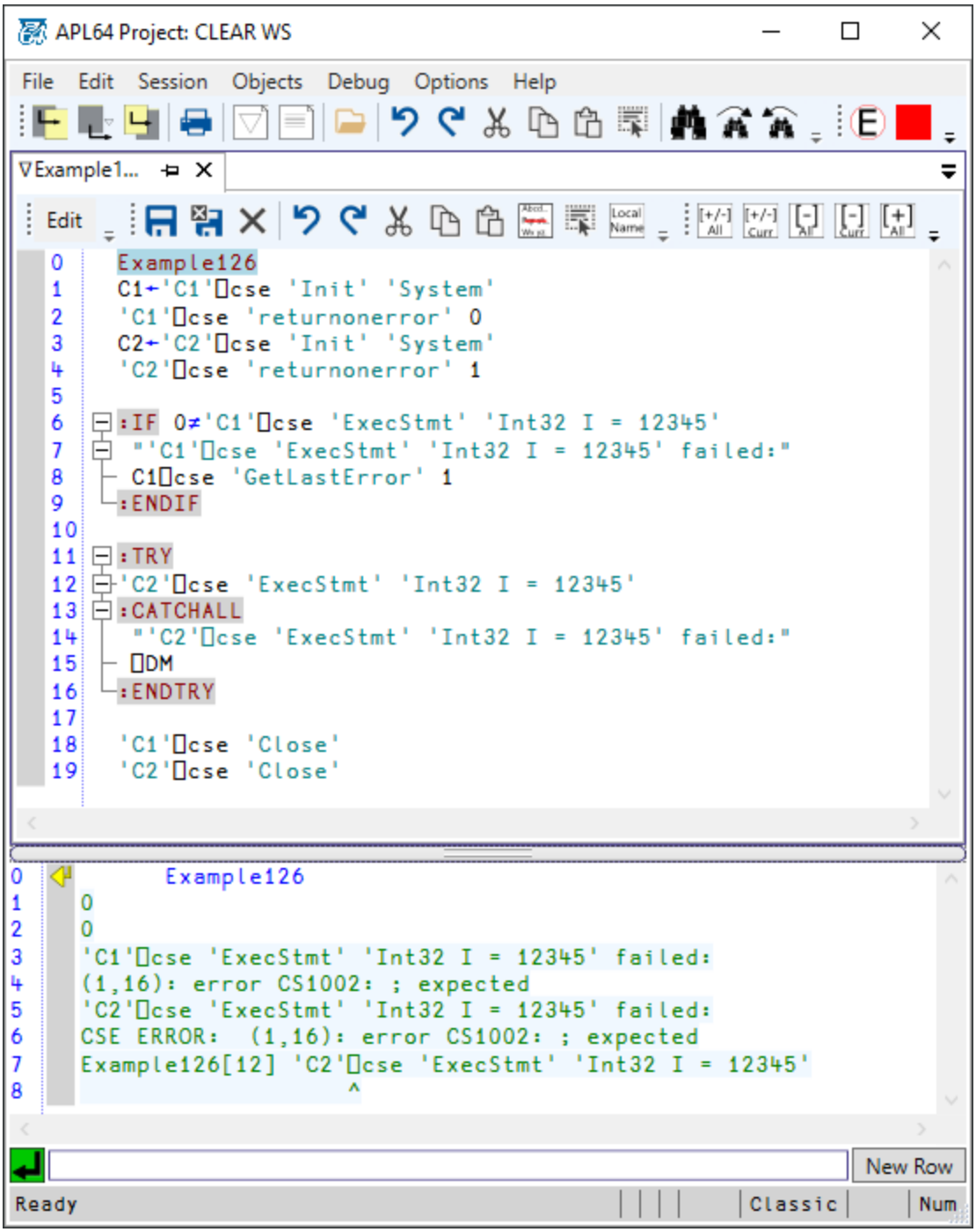
```
Example125
□cself←'cse'□cse 'Init' 'System'
□cse 'ReturnOnError'
□cse 'ReturnOnError' 1
□cse 'ReturnOnError'
□cse 'Close'
```


APL64 Project: CLEAR WS
File Edit Session Objects Debug Options Help
Example1...
Edit
0 Example125
1 ⎕cself←'cse'⎕cse 'Init' 'System'
2 ⎕cse 'ReturnOnError'
3 ⎕cse 'ReturnOnError' 1
4 ⎕cse 'ReturnOnError'
5 ⎕cse 'Close'
0 Example125
1 0
2 0
3 1
New Row
Ready
Classic
Num

Example #126:

The value and action of the CSE ‘returnonerror’ property is local to an instance of the CSE object. In this example the action of the system when the CSE ‘returnonerror’ property is 0 or 1 is illustrated:

```
Example126
C1←'C1'⎕cse 'Init' 'System'
'C1'⎕cse 'returnonerror' 0
C2←'C2'⎕cse 'Init' 'System'
'C2'⎕cse 'returnonerror' 1

:IF 0≠'C1'⎕cse 'ExecStmt' 'Int32 I = 12345'
 "'C1'⎕cse 'ExecStmt' 'Int32 I = 12345' failed:"
 C1⎕cse 'GetLastError' 1
:ENDIF

:TRY
'C2'⎕cse 'ExecStmt' 'Int32 I = 12345'
:CATCHALL
 "'C2'⎕cse 'ExecStmt' 'Int32 I = 12345' failed:"
 ⎕DM
:ENDTRY

'C1'⎕cse 'Close'
'C2'⎕cse 'Close'
```



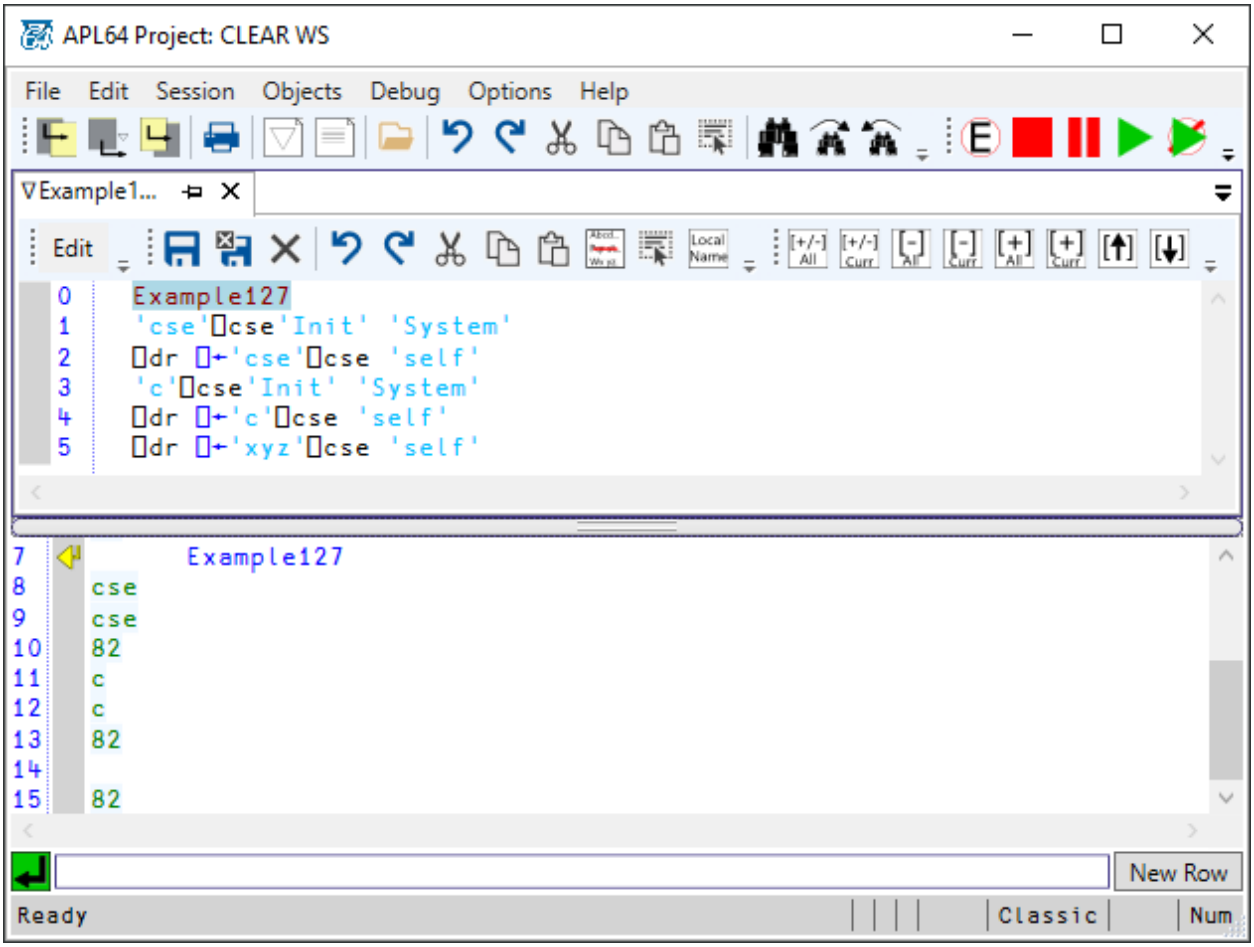
CSE self property

The CSE ‘self’ property is a read-only property which returns the APL64 name associated with an instance of the CSE object if that object currently exists or '' (i.e. 0ρ' ') if there is no CSE object associated with the name. The value of the CSE ‘self’ property is a character scalar or vector.

Example #127:

```
Example127
```

```
'cse'□cse'Init' 'System
□dr □←'cse'□cse 'self'
'c'□cse'Init' 'System
□dr □←'c'□cse 'self'
□dr □←'xyz'□cse 'self'
```



CSE trackevents Property

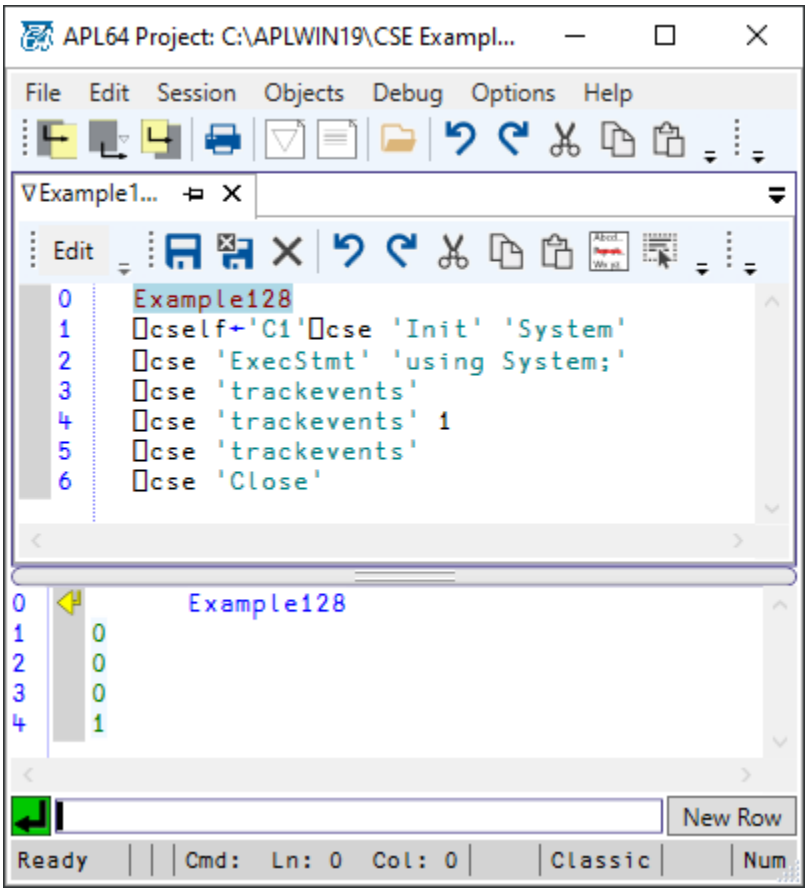
The CSE 'trackevents' property provides a way to have information, about the firing of C# events which have been subscribed via the CSE, logged to the Windows Application event log, so that they can be seen using the Windows Event Viewer. This feature is useful when testing an APL64 application system which has subscribed to a C# event using the CSE 'AddEventHandler' method.

The right argument may be set to 1 (indicating true) or 0 (indicating false). The default value of this CSE property is 0.

When the optional right argument is set to 1, events which have been subscribed by APL64 and have fired will have messages sent to the Windows Application event log. Routing of event firings to the APL64 calling environment are not affected by the value of this property.

Example #128:
When setting the value of the CSE 'trackevents' property the prior value of this property will be returned.

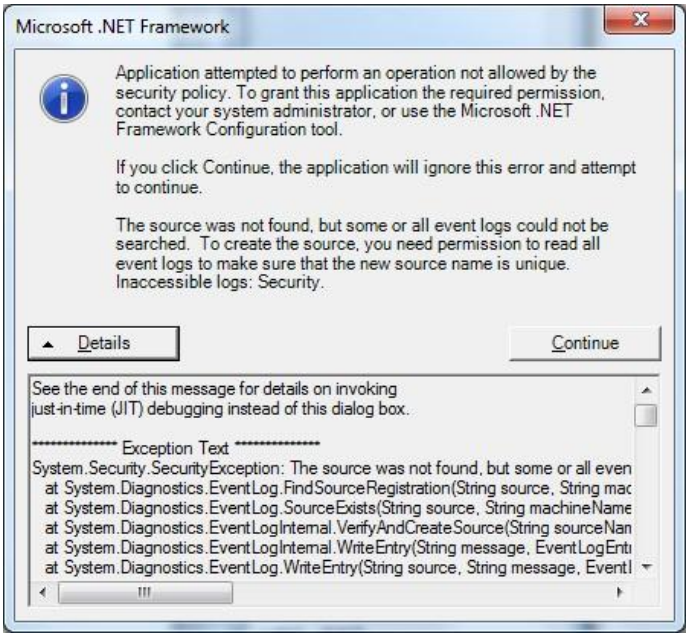
```
Example128
 cself←'C1' cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'
 cse 'trackevents'
 cse 'trackevents' 1
 cse 'trackevents'
 cse 'Close'
```



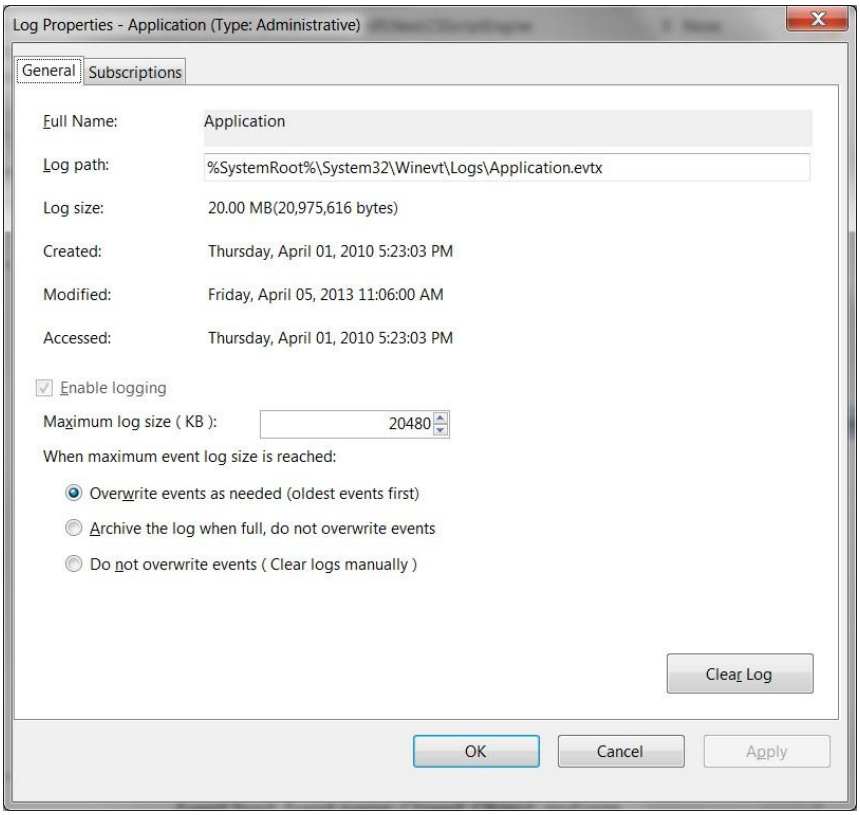
It is not recommended that the value of this property be set to 1 in a production environment because:

- User-controlled settings on the Windows Application event log may cause unexpected errors.
- Posting messages to the Windows Application event log consumes processing resources
- The CSE 'trackevents' property will attempt to write to the Windows Application event log, so the APL64 session must be 'Run As Administrator'.

If the APL64 session is not 'Run As Administrator' and the 'trackevents' property value is set to 1, a 'Security Policy Error' exception will be signaled when a CSE event fires and the CSE attempts to write to the Windows Application event log, e.g.:



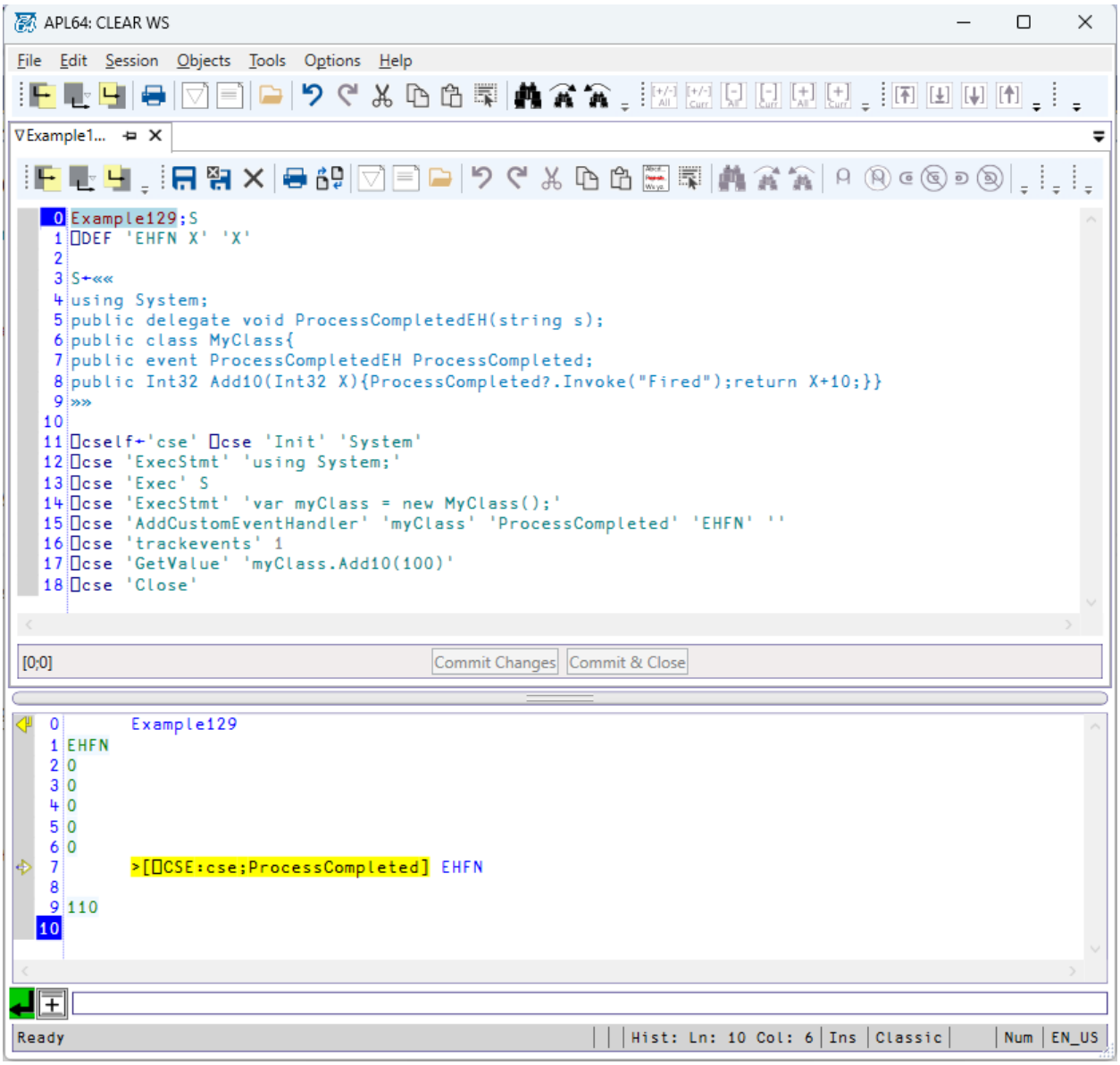
Besides running the APL64 session 'As Administrator', be sure that the Windows Application event log properties are set so that events can be written to it:



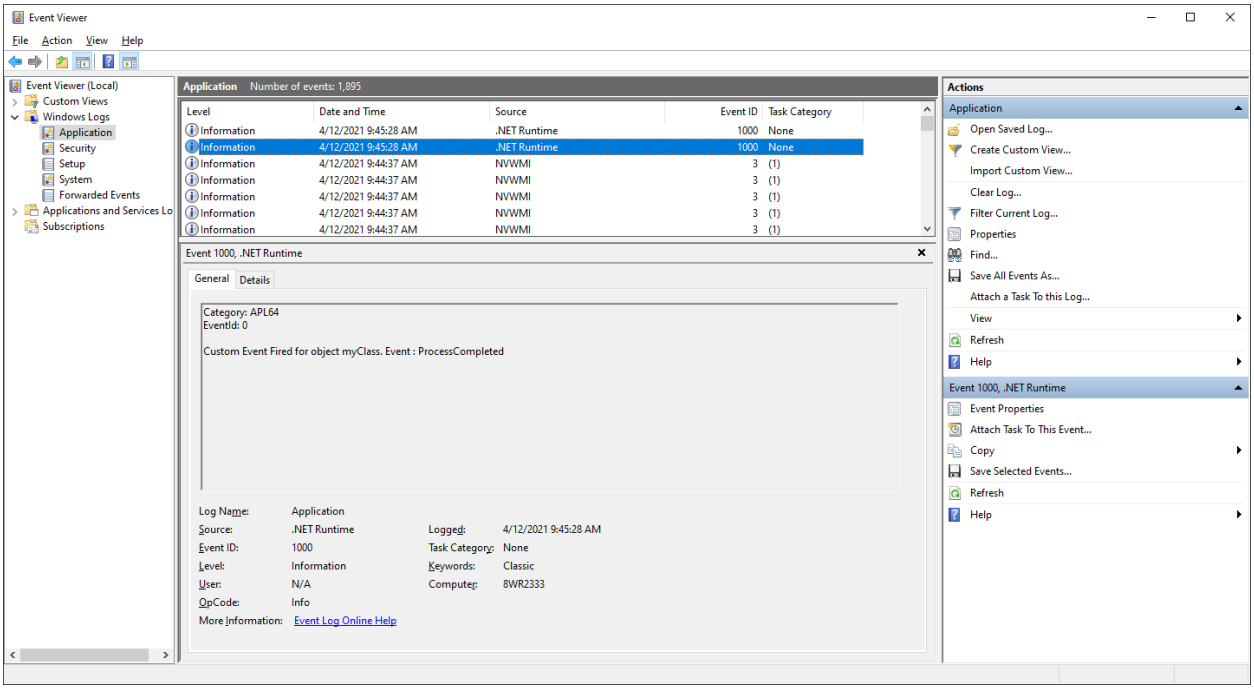
Example #129:

In this example the CSE 'trackevents' property is set to 1. The MyClass.Add10() method exposes the ProcessCompleted event which is subscribed by APL64 using the CSE. When this method is used, the ProcessCompleted event is fired and handled by the programmer-defined EHFN function and the Windows Application event log is updated with the event firing information.

```
Example129;S
⍝DEF 'EHFN X' 'X'

S←««
using System;
public delegate void ProcessCompletedEH(string s);
public class MyClass{
public event ProcessCompletedEH ProcessCompleted;
public Int32 Add10(Int32 X){ProcessCompleted?.Invoke("Fired");return X+10;}}
»»

⍝cself←'cse' ⍝cse 'Init' 'System'
⍝cse 'ExecStmt' 'using System;'
⍝cse 'Exec' S
⍝cse 'ExecStmt' 'var myClass = new MyClass();'
⍝cse 'AddCustomEventHandler' 'myClass' 'ProcessCompleted' 'EHFN' ''
⍝cse 'trackevents' 1
⍝cse 'GetValue' 'myClass.Add10(100)'
⍝cse 'Close'
```



The event is logged in the Windows Event Log:



Maximizing the benefits of the CSE in an APL64 Application

.Net has been continually enhanced since Microsoft made it publicly available in 2002.
Many third-party software development organizations have created additional components for .Net. .Net is better organized and documented than any prior implementation of the Microsoft Windows application programming interface (API). Often .Net features can be used to save design and programming time so that an APL64 application system can be delivered faster with improved features and reduced cost.

To get the most out of the CSE and the Microsoft .Net it is necessary to learn about the C# programming language. Numerous tutorials and books are available to accomplish this goal.

- Web search "Learn C#"
- Microsoft Documentation
- http://www.apress.com/microsoft/c
- http://stackoverflow.com/

APL2000 consultants are ready to provide specific help about using C#, .Net and the APL64 □cse system function. Contact sales@apl2000.com for more information.

Future of the CSE

APL2000 customer feedback about the CSE is encouraged and will direct future development of the CSE facility in APL64.

Proprietary Information

This documentation contains proprietary information of APLNext LLC, including trade secrets and confidential information, under license to APLNow LLC. Use of this information is governed by the APL64 End User License Agreement (EULA).

Developing and deploying .Net Assemblies for use with the CSE

Sometimes it is desirable to develop and deploy a custom .Net assembly for use with the CSE. Microsoft Visual Studio is the ideal tool for this development effort. Such .Net assemblies may be loaded into a CSE instance using the CSE 'LoadAssembly' method.

Detailed CSE Examples

This section assumes that the applicable APL64 and APLNext C# Script Engine versions have been installed to the APL64 programmer's workstation.

These examples may illustrate multiple sequential uses of the CSE 'ExecStmt' method. This is done to illustrate how the CSE can assist in developing and testing a new CSE script in a line-by-line manner. After testing is complete, the C# statements can be gathered into:

- A CSE script in an APL64 rank-2 text array executed using the CSE 'Exec' method, or
- A CSE script file executed using the CSE 'ExecFile' method, or
- A .Net assembly compiled using Visual Studio and then loaded using the CSE 'LoadAssembly' or 'LoadAssemblyByName' methods.

Because the CSE is designed to use native C# source code, the universe of C# examples using the Microsoft .Net that is available via the Internet becomes a readily-available source of tools for the APL64 programmer who investigates and masters the CSE.

In some of these examples the use of proprietary software components is illustrated. APLNext, LLC has no business relationship with suppliers of this proprietary software. By their inclusion in this documentation, no representation is being made as to the suitability for any purpose of these components.

These examples are not intended to illustrate all possibilities or options available with the CSE and APL64. The source code provided in the example, if any, is generally not suitable for production use because:

- Application-specific modifications may be necessary
- Exception handling necessary for a robust application system needs to be incorporated
- Other options available in APL64 or C# may be more suitable

Creating C Sharp Classes Methods Properties & Fields

In an instance of the CSE object it is possible to create C# classes, methods, properties and fields. The following examples illustrate these possibilities.

A C# field or variable is defined by its type, name and value. Stylistically a C# field is generally not exposed as a public object of a C# class.

A C# property provides more flexibility for the programmer and can be used in more circumstances. A C# property is often exposed as a public object of a C# class. A C# property is actually a pair of associated C# methods to get and set the value of the property. A C# property is distinct from a C# field.

A C# method is analogous to an APL64 function. A C# method has a specified result type and argument type(s).

A C# class is a .Net object that is a container for the definitions of C# fields, properties and methods.

An instance of the CSE object can contain methods, properties and fields which are not defined within a C# class. A CSE instance can contain multiple C# class, property and method definitions. It is possible to have multiple C# methods with the same name, called 'overloads', however the argument structure of each overload of the same-named method must be distinct.

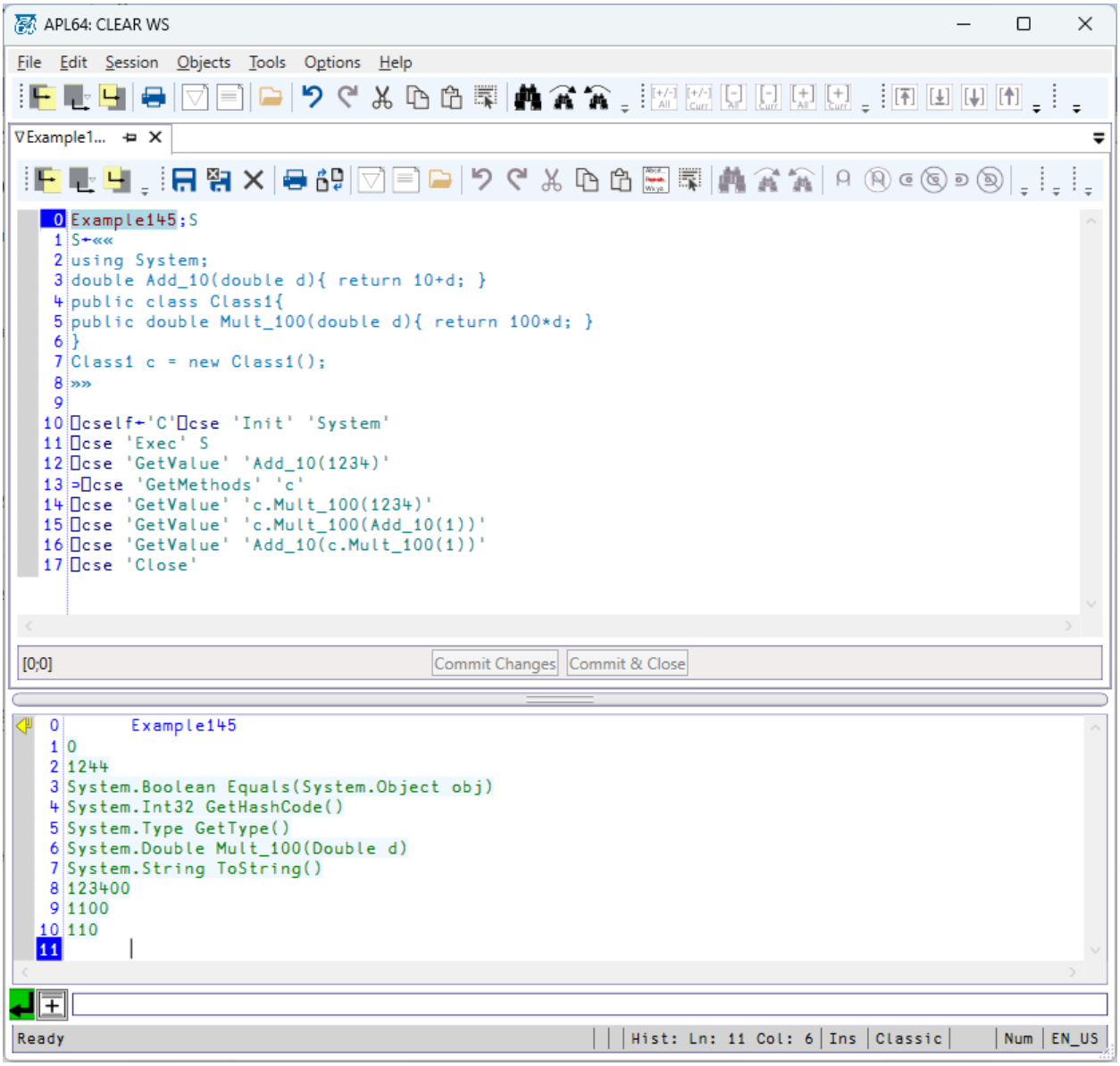
Example #145:
In this example two C# methods are defined:

- The 'Add_10' C# method is defined without 'enclosing' it a .Net class
- The 'Mult_100' C# method is defined in the 'MyClass' C# class

Because the 'Add_10' method is defined without an 'enclosing' class it is deemed public and therefore available to any object which exists in the instance of the CSE object. Because the 'Mult_100' method is defined within the 'enclosing' class 'MyClass', it is public, but only when referenced as a method of an instance of the MyClass type.

```
Example145;S
S←««
using System;
double Add_10(double d){ return 10+d; }
public class Class1{
public double Mult_100(double d){ return 100*d; }
}
Class1 c = new Class1();
»»

□cself←'C'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'GetValue' 'Add_10(1234)'
⊃□cse 'GetMethods' 'c'
□cse 'GetValue' 'c.Mult_100(1234)'
□cse 'GetValue' 'c.Mult_100(Add_10(1))'
```

```
]cse 'GetValue' 'Add_10(c.Mult_100(1))'
]cse 'Close'
```



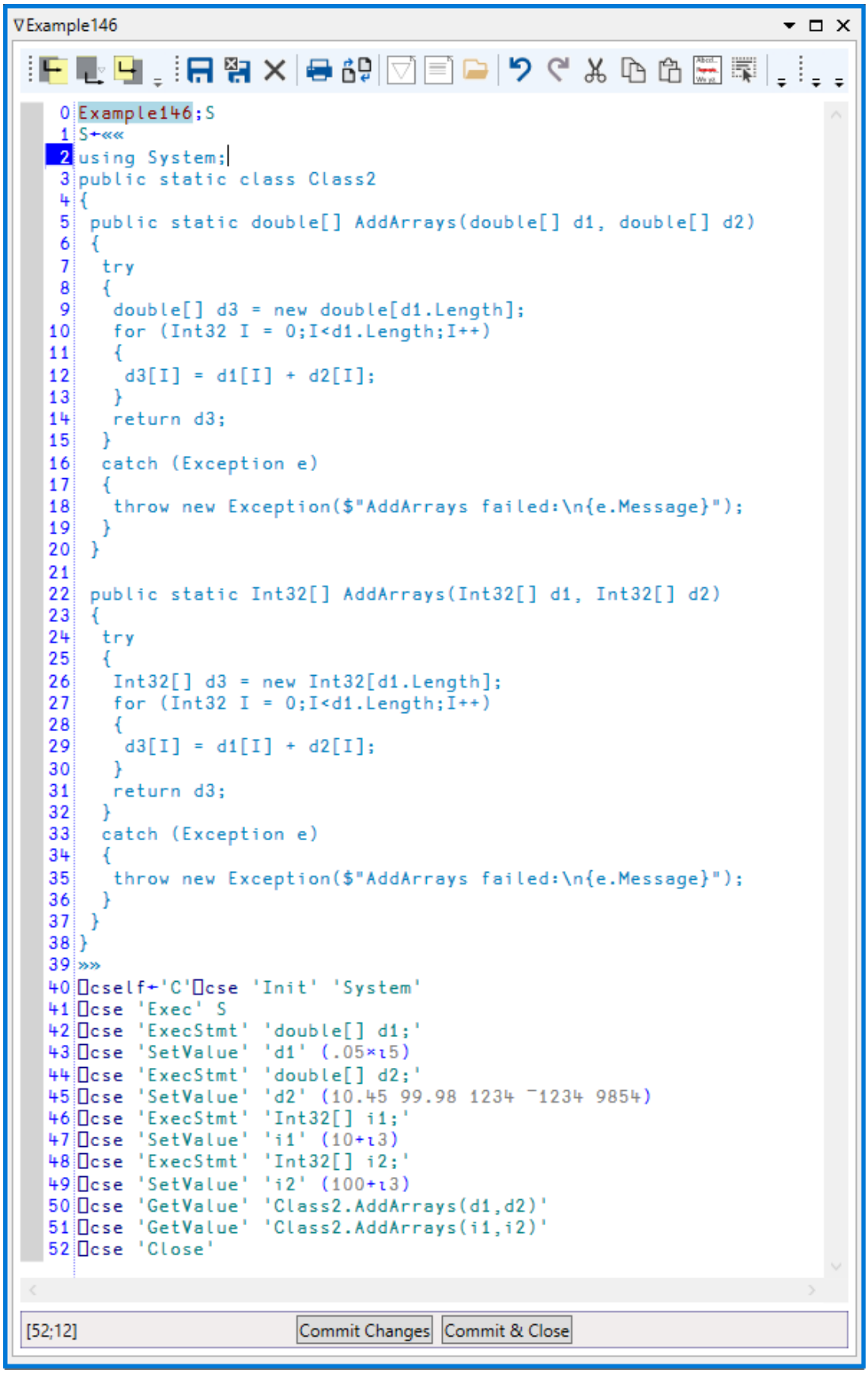
Example #146:

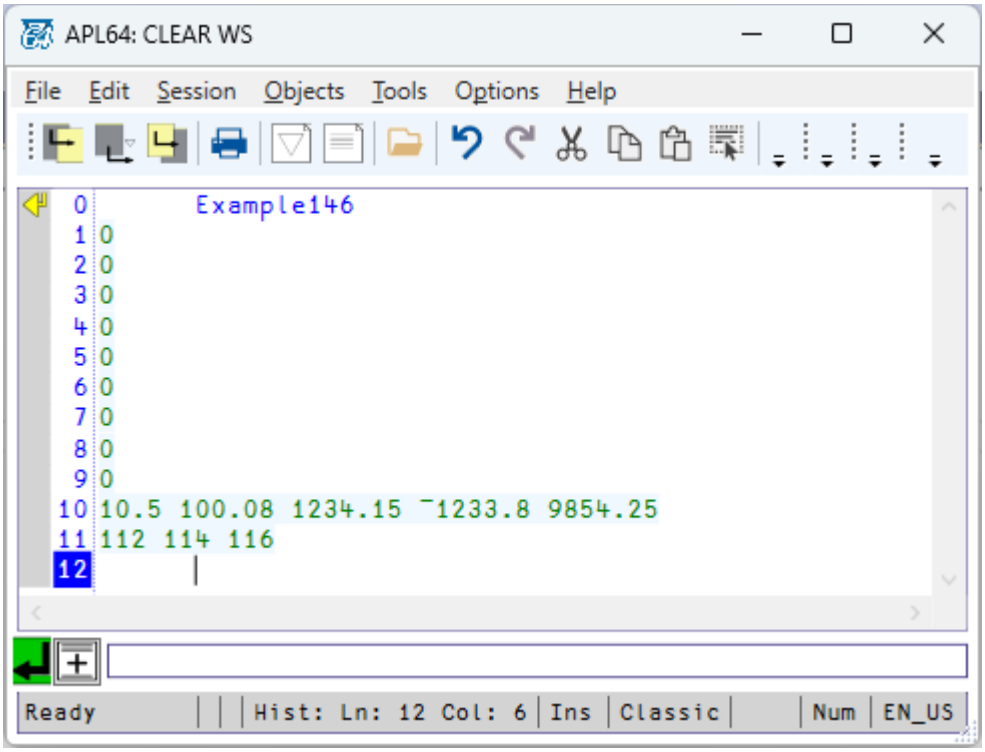
In this example in an instance of the CSE a C# static class is defined and two C# static methods are defined in that class that are overloads of each other. Because the class and methods are static, the methods are accessed using the class name prefix. The argument structure of the two overloads is different since one overload takes an argument of C# type double[] and the other overload takes an argument of C# type Int32[].

```
Example146;S
S←««
using System;
```

```
public static class Class2
{
 public static double[] AddArrays(double[] d1, double[] d2)
 {
  try
  {
   double[] d3 = new double[d1.Length];
   for (Int32 I = 0;I<d1.Length;I++)
   {
    d3[I] = d1[I] + d2[I];
   }
   return d3;
  }
  catch (Exception e)
  {
   throw new Exception($"AddArrays failed:\n{e.Message}");
  }
 }

 public static Int32[] AddArrays(Int32[] d1, Int32[] d2)
 {
  try
  {
   Int32[] d3 = new Int32[d1.Length];
   for (Int32 I = 0;I<d1.Length;I++)
   {
    d3[I] = d1[I] + d2[I];
   }
   return d3;
  }
  catch (Exception e)
  {
   throw new Exception($"AddArrays failed:\n{e.Message}");
  }
 }
}
»»
□cself←'C'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'ExecStmt' 'double[] d1;'
□cse 'SetValue' 'd1' (.05×⍳5)
□cse 'ExecStmt' 'double[] d2;'
□cse 'SetValue' 'd2' (10.45 99.98 1234 ¯1234 9854)
```

```
⎕cse 'ExecStmt' 'Int32[] i1;'
⎕cse 'SetValue' 'i1' (10+⍳3)
⎕cse 'ExecStmt' 'Int32[] i2;'
⎕cse 'SetValue' 'i2' (100+⍳3)
⎕cse 'GetValue' 'Class2.AddArrays(d1,d2)'
⎕cse 'GetValue' 'Class2.AddArrays(i1,i2)'
⎕cse 'Close'
```


VExample146
0 Example146;S
1 S←⍬⍬
2 using System;
3 public static class Class2
4 {
5 public static double[] AddArrays(double[] d1, double[] d2)
6 {
7 try
8 {
9 double[] d3 = new double[d1.Length];
10 for (Int32 I = 0;I<d1.Length;I++)
11 {
12 d3[I] = d1[I] + d2[I];
13 }
14 return d3;
15 }
16 catch (Exception e)
17 {
18 throw new Exception($"AddArrays failed:\n{e.Message}");
19 }
20 }
21
22 public static Int32[] AddArrays(Int32[] d1, Int32[] d2)
23 {
24 try
25 {
26 Int32[] d3 = new Int32[d1.Length];
27 for (Int32 I = 0;I<d1.Length;I++)
28 {
29 d3[I] = d1[I] + d2[I];
30 }
31 return d3;
32 }
33 catch (Exception e)
34 {
35 throw new Exception($"AddArrays failed:\n{e.Message}");
36 }
37 }
38 }
39 ⍝»»
40 ⎕cself←'C'⎕cse 'Init' 'System'
41 ⎕cse 'Exec' S
42 ⎕cse 'ExecStmt' 'double[] d1;'
43 ⎕cse 'SetValue' 'd1' (.05×⍳5)
44 ⎕cse 'ExecStmt' 'double[] d2;'
45 ⎕cse 'SetValue' 'd2' (10.45 99.98 1234 ¯1234 9854)
46 ⎕cse 'ExecStmt' 'Int32[] i1;'
47 ⎕cse 'SetValue' 'i1' (10+⍳3)
48 ⎕cse 'ExecStmt' 'Int32[] i2;'
49 ⎕cse 'SetValue' 'i2' (100+⍳3)
50 ⎕cse 'GetValue' 'Class2.AddArrays(d1,d2)'
51 ⎕cse 'GetValue' 'Class2.AddArrays(i1,i2)'
52 ⎕cse 'Close'
[52;12]
Commit Changes
Commit & Close

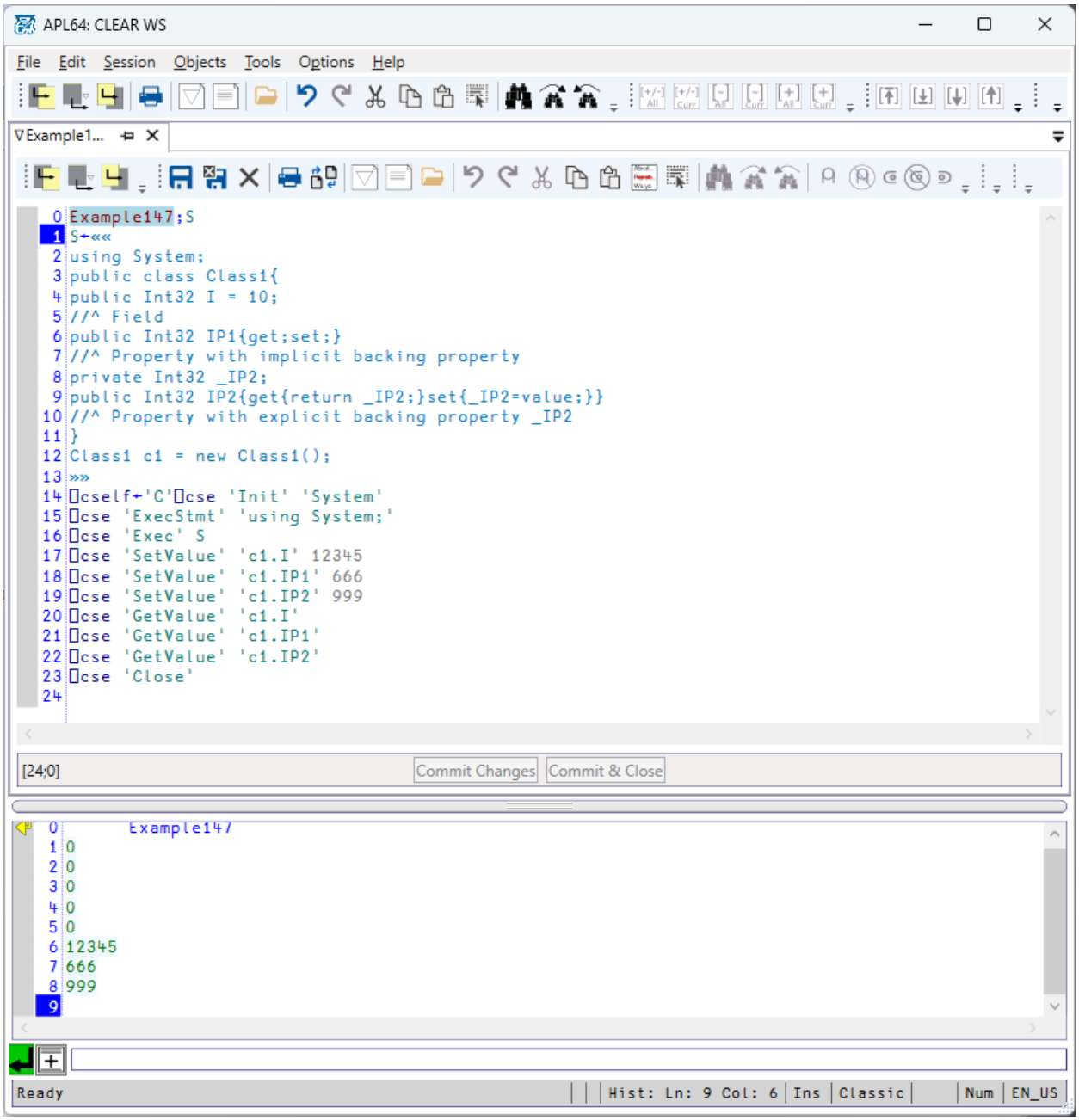


Example #147:
In this example the C# class ‘Class1’ includes the definitions of:

- The C# field ‘I’
- The C# properties ‘IP1’ and ‘IP2

```
Example147;S
S←««
using System;
public class Class1{
public Int32 I = 10;
//^ Field
public Int32 IP1{get;set;}
//^ Property with implicit backing property
private Int32 _IP2;
public Int32 IP2{get{return _IP2;}set{_IP2=value;}}
//^ Property with explicit backing property _IP2
}
Class1 c1 = new Class1();
»»
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'Exec' S
```

- [ ] cse 'SetValue' 'c1.I' 12345
- [ ] cse 'SetValue' 'c1.IP1' 666
- [ ] cse 'SetValue' 'c1.IP2' 999
- [ ] cse 'GetValue' 'c1.I'
- [ ] cse 'GetValue' 'c1.IP1'
- [ ] cse 'GetValue' 'c1.IP2'
- [ ] cse 'Close'

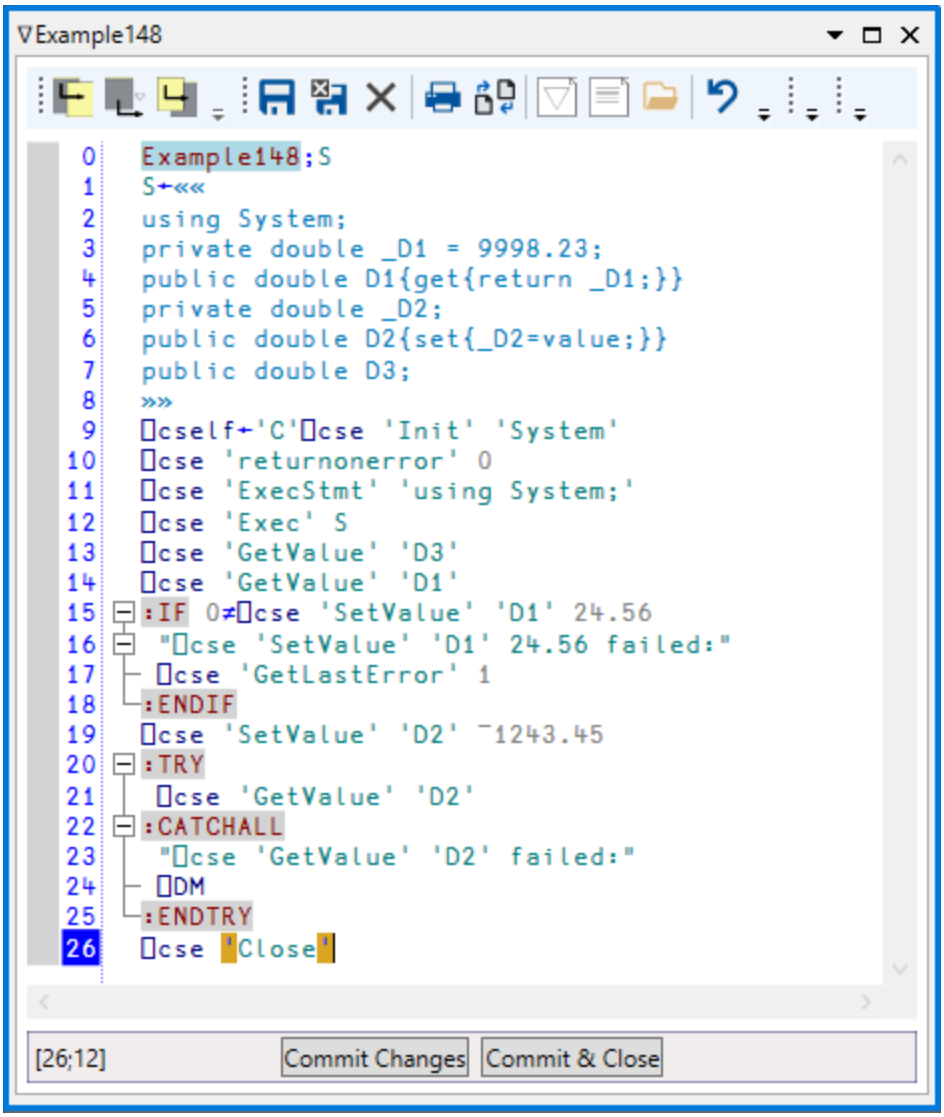


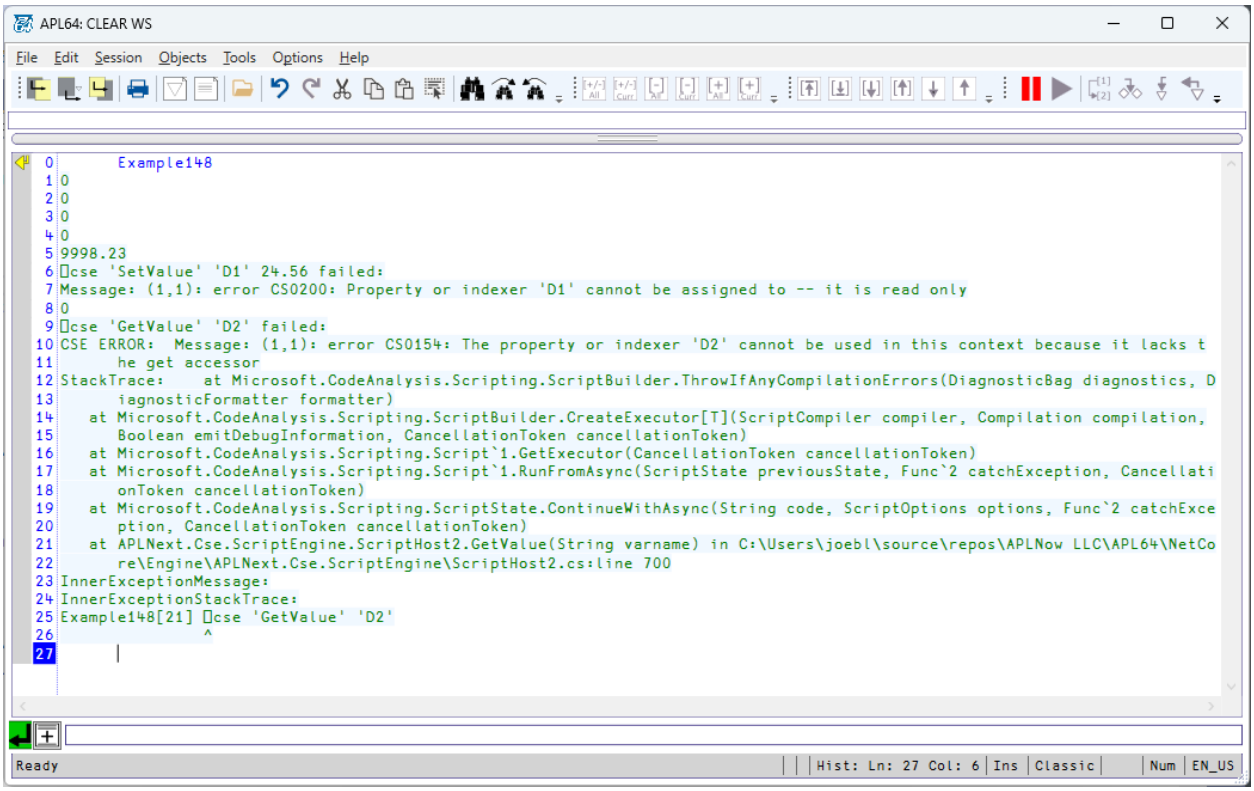
Example #148:

In this example the CSE script ‘S’ defines:

- The C# double field ‘D3’ without explicitly specifying its value. The CSE ‘GetValue’ method returns 0 which is the default value for a .Net double.
- The C# double property D1 with a ‘get’, but no ‘set’ accessor method. The CSE ‘GetValue’ method returns the default double value of 0, but the CSE ‘SetValue’ method causes C# to throw an exception because there is no ‘set ‘ accessor.
- The C# double property D2 with no explicit ‘get’, but with a ‘set’ accessor method. The CSE ‘SetValue’ method causes C# to throw an exception because there is no ‘get’ accessor.

```
Example148;S
S←««
using System;
private double _D1 = 9998.23;
public double D1{get{return _D1;}}
private double _D2;
public double D2{set{_D2=value;}}
public double D3;
»»
⎕←cself←'C'⎕cse 'Init' 'System'
⎕←cse 'returnonerror' 0
⎕←cse 'ExecStmt' 'using System;'
⎕←cse 'Exec' S
⎕←cse 'GetValue' 'D3'
⎕←cse 'GetValue' 'D1'
:IF 0≠⎕←cse 'SetValue' 'D1' 24.56
 "⎕←cse 'SetValue' 'D1' 24.56 failed:"
 ⎕←cse 'GetLastError' 1
:ENDIF
⎕←cse 'SetValue' 'D2' ¯1243.45
:TRY
 ⎕←cse 'GetValue' 'D2'
:CATCHALL
 "⎕←cse 'GetValue' 'D2' failed:"
 ⎕←DM
:ENDTRY
⎕←cse 'Close'
```


Example148
0 Example148;S
1 S←⍬⍬
2 using System;
3 private double _D1 = 9998.23;
4 public double D1{get{return _D1;}}
5 private double _D2;
6 public double D2{set{_D2=value;}}
7 public double D3;
8 ⍬⍬
9 ⎕cself←'C'⎕cse 'Init' 'System'
10 ⎕cse 'returnonerror' 0
11 ⎕cse 'ExecStmt' 'using System;'
12 ⎕cse 'Exec' S
13 ⎕cse 'GetValue' 'D3'
14 ⎕cse 'GetValue' 'D1'
15 :IF 0≠⎕cse 'SetValue' 'D1' 24.56
16 "⎕cse 'SetValue' 'D1' 24.56 failed:"
17 ⎕cse 'GetLastError' 1
18 :ENDIF
19 ⎕cse 'SetValue' 'D2' ¯1243.45
20 :TRY
21 ⎕cse 'GetValue' 'D2'
22 :CATCHALL
23 "⎕cse 'GetValue' 'D2' failed:"
24 ⎕DM
25 :ENDTRY
26 ⎕cse 'Close'
[26;12]
Commit Changes
Commit & Close



Example #172:

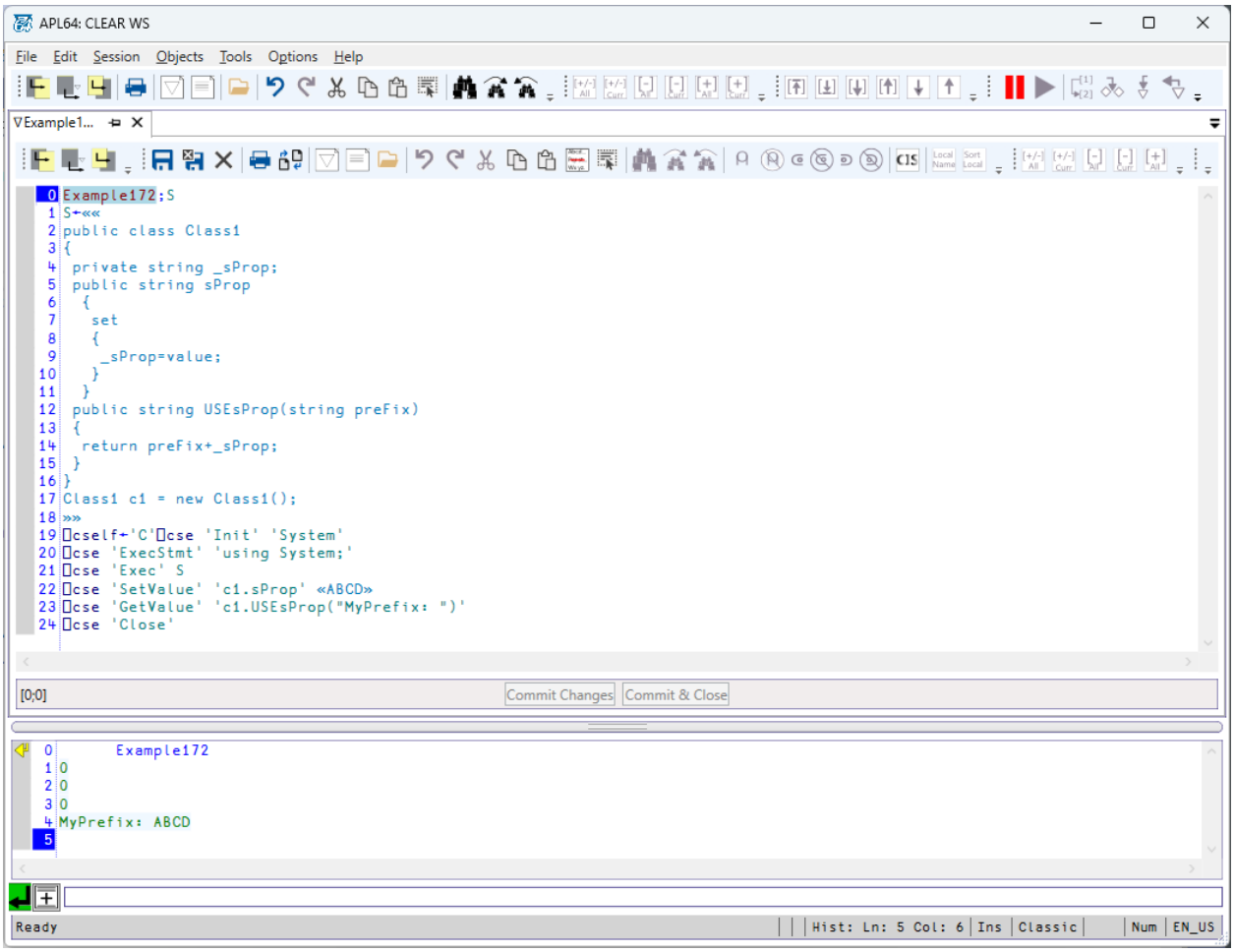
In this example the CSE script ‘Script172’ defines and creates an instance ‘c1’ of the class ‘Class1’ with

- A public property ‘sProp’ with a setter
- A private ‘back’ property ‘_sProp’
- A method ‘USEsProp’ which uses _sProp

The ‘Example172’ function uses the class instance ‘c1’ to set the value of the ‘sProp’ property and execute the ‘USEsProp’ method.

```
Example172;S
S←««
public class Class1
{
 private string _sProp;
 public string sProp
  {
   set
   {
    _sProp=value;
   }
  }
 public string USEsProp(string preFix)
 {
```

```
  return preFix+_sProp;
 }
}
Class1 c1 = new Class1();
»»
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'Exec' S
□cse 'SetValue' 'c1.sProp' «ABCD»
□cse 'GetValue' 'c1.USEsProp("MyPrefix: ")'
□cse 'Close'
```



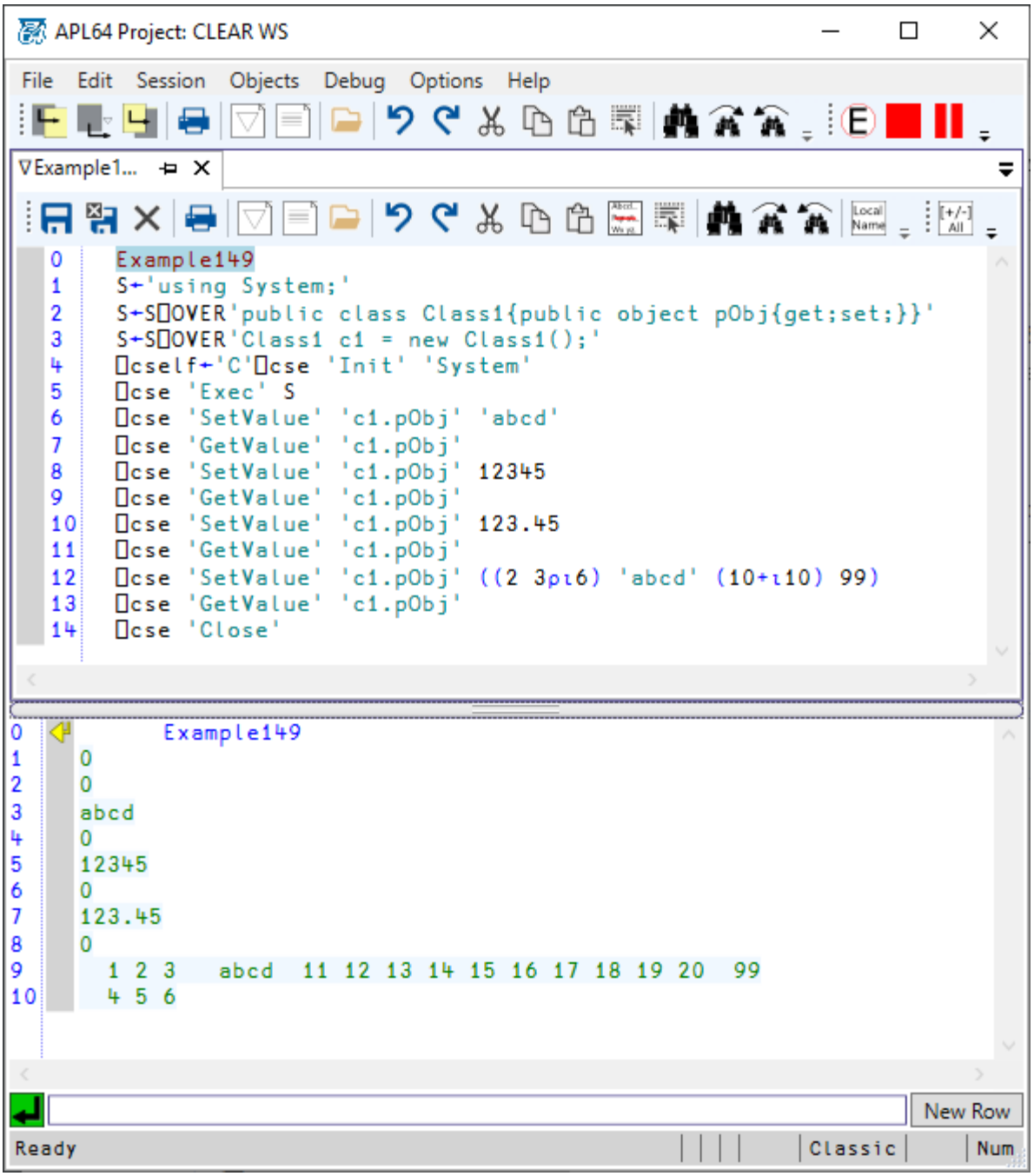
.Net Class with a Property of Type Object

It is possible to define a class with a property of type object and use the CSE 'GetValue' and 'SetValue' methods to modify and retrieve the value of this property. The object type is the base class of all .Net objects, so this means that the 'implied' type of this property can be changed using CSE technology.

Example #149:

In this example the instance class ‘Class1’ and the pObj property are defined in the CSE script. When an instance of the Class1 class is created, the pObj property values can be set and retrieved by the CSE ‘SetValue’ and ‘GetValue’ methods. Because the defined type of the pObj property of the Class1 class is object, its value can be set to any data type which APL64 can send.

```
Example149
S←'using System;'
S←S□OVER'public class Class1{public object pObj{get;set;}}'
S←S□OVER'Class1 c1 = new Class1();'
□cself←'C'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'SetValue' 'c1.pObj' 'abcd'
□cse 'SetValue' 'c1.pObj' «abcd»
□cse 'GetValue' 'c1.pObj'□cse 'SetValue' 'c1.pObj' 12345
□cse 'GetValue' 'c1.pObj'
□cse 'SetValue' 'c1.pObj' 123.45
□cse 'GetValue' 'c1.pObj'
□cse 'SetValue' 'c1.pObj' ((2 3⍴⍳6) 'abcd' (10+⍳10) 99)
□cse 'GetValue' 'c1.pObj'
□cse 'Close'
```



Using JSON with APL64 via the CSE

Example #155:

JSON (Javascript Object Notation) is a less verbose serialization compared to XML serialization. Several .Net assemblies are available to parse, query and construct JSON data. This example uses the Newtonsoft Json open source .Net package.

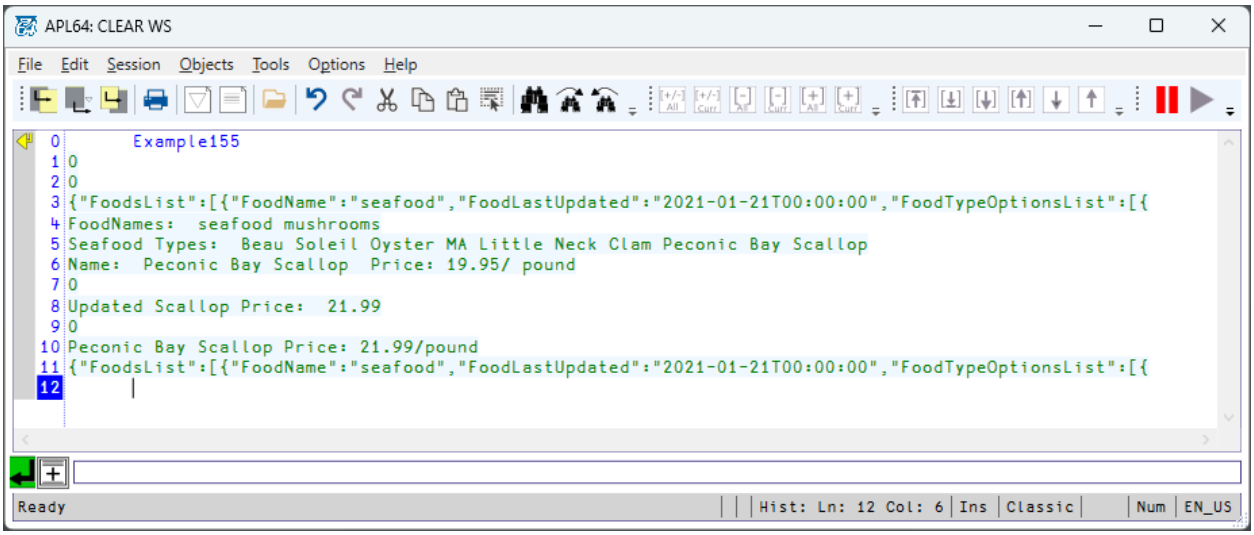
In this example sample Json format data is created, accessed and updated. Using .Net Linq and properties, the Json format data can be queried and updated. In this example three CSE scripts are used to create the CSE instance features.

```
Example155;S
⍞←cself←'C' ⍞←cse 'Init' 'System' 'System.Text.Json'

S←««
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
public class FoodTypeOption
{
 public string FoodTypeName {get; set; }
 public string FoodTypeOrigin {get; set;}
 public string FoodTypeUnit {get; set;}
 public double FoodTypePrice {get; set;}
}
public class FoodTypeOptions
{
 public string FoodName {get; set;}
 public DateTime FoodLastUpdated {get;set;}
 public List<FoodTypeOption> FoodTypeOptionsList {get; set;} = new List<FoodTypeOption>();
}
public class Foods
{
 public List<FoodTypeOptions> FoodsList {get; set;} = new List<FoodTypeOptions>();
 public static Foods DeSerializeFoodsFromJson(string foodsString)
 {
  return JsonSerializer.Deserialize<Foods>(foodsString);
 }
 public static string SerializeFoodsAsJson(Foods foods)
 {
  return JsonSerializer.Serialize(foods);
 }
}
»»
⍞←cse 'Exec' S
⍝↑ Create .Net classes and json serialize/deserialize methods for Foods class
⍝  The .Net Linq tools are now available
```

```
S←««
var foods = new Foods()
{
   FoodsList = new List<FoodTypeOptions>()
   {
      new FoodTypeOptions()
      {
         FoodName = "seafood",
         FoodLastUpdated = DateTime.Parse("2021-01-21T00:00:00"),
         FoodTypeOptionsList = new List<FoodTypeOption>()
         {
            new FoodTypeOption()
            {
               FoodTypeName = "Beau Soleil Oyster",
               FoodTypeOrigin = "Neguac, NB, Canada",
               FoodTypeUnit = "each",
               FoodTypePrice = 0.75
            },
            new FoodTypeOption()
            {
               FoodTypeName = "MA Little Neck Clam",
               FoodTypeOrigin = "Nauset Inlet, MA, USA",
               FoodTypeUnit = "each",
               FoodTypePrice = 0.56
            },
            new FoodTypeOption()
            {
               FoodTypeName = "Peconic Bay Scallop",
               FoodTypeOrigin = "Peconic Bay, NY, USA",
               FoodTypeUnit = "pound",
               FoodTypePrice = 19.95
            }
         }
      },
      new FoodTypeOptions()
      {
         FoodName = "mushrooms",
         FoodLastUpdated = DateTime.Parse("2021-01-31T00:00:00"),
         FoodTypeOptionsList = new List<FoodTypeOption>()
         {
            new FoodTypeOption()
```

```
          {
            FoodTypeName = "Italian Porcini",
            FoodTypeOrigin = "Puglia, IT",
            FoodTypeUnit = "gram",
            FoodTypePrice = 11.22
          },
          new FoodTypeOption()
          {
            FoodTypeName = "Polish Pfefferlinge",
            FoodTypeOrigin = "Wroclaw, PL",
            FoodTypeUnit = "gram",
            FoodTypePrice = 7.56
          },
          new FoodTypeOption()
          {
            FoodTypeName = "Washington State Morel",
            FoodTypeOrigin = "Seattle, WA, USA",
            FoodTypeUnit = "ounce",
            FoodTypePrice = 22.95
          }
        }
      }
    }
};
»»
⎕cse 'Exec' S
⍝↑ Create a Foods class instance with sample data

100↑>jsonFoods←⎕cse 'GetValue' 'Foods.SerializeFoodsAsJson(foods)'
⍝↑ Get the json-serialized foods info as a string
⍝  Display first 100 characters

'FoodNames: ', ⎕cse 'GetValue' '(from e in foods.FoodsList select e.FoodName).ToArray()'
⍝↑ Use .Net Linq and SQL language

'Seafood Types: ', ⎕cse 'GetValue' 'foods.FoodsList[0].FoodTypeOptionsList.Select(elt =>
elt.FoodTypeName).ToArray()'
⍝↑ Use indexing on the FoodsList collection and then link with lambda expression

←⎕cse 'ExecStmt' 'var scallop = foods.FoodsList[0].FoodTypeOptionsList[2];'
⍝↑ Define scallop object using indices on foods object and its children
scallopInfo←⎕cse 'GetValue' 'scallop.FoodTypeName'
```

```
scallopInfo←scallopInfo,⎕cse 'GetValue' 'scallop.FoodTypePrice'
scallopInfo←scallopInfo,⎕cse 'GetValue' 'scallop.FoodTypeUnit'
'Name: ',scallopInfo[1],' Price: ',(⍕scallopInfo[2]),'/',scallopInfo[3]
⎕cse 'ExecStmt' 'scallop.FoodTypePrice = {0}' 21.99
⍝↑ Update the price of scallop
'Updated Scallop Price: ',⎕cse 'GetValue' 'scallop.FoodTypePrice'

S←««
public string GetScallopInfo()
{
 var res = String.Empty;
 var foodTypeOptions = foods.FoodsList.First(e => e.FoodName == "seafood");
 if (foodTypeOptions != null)
 {
  var foodTypeOption = foodTypeOptions.FoodTypeOptionsList.First(e => e.FoodTypeName == "Peconic
Bay Scallop");
  if (foodTypeOption != null)
  {
   res = $"{foodTypeOption.FoodTypeName} Price:
{foodTypeOption.FoodTypePrice}/{foodTypeOption.FoodTypeUnit}";
  }
 }
 return res;
}
»»
⍝↑ CSE Script to obtain scallop info using linq and lambda expressions
⎕cse 'Exec' S

⎕cse 'GetValue' 'GetScallopInfo()'

100↑>jsonFoodsUpdated←⎕cse'GetValue' 'Foods.SerializeFoodsAsJson(foods)'
⍝↑ Get json-serialize updated foods info
⍝  Display first 100 characters


⎕cse 'Close'
```



The json format sample data in Notepad++:


*C:\Junk\FoodsList.json - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
FoodsList.json
"FoodsList": [
{
"FoodName": "seafood",
"FoodLastUpdated": "2021-01-21T00:00:00",
"FoodTypeOptionsList": [
{
"FoodTypeName": "Beau Soleil Oyster",
"FoodTypeOrigin": "Neguac, NB, Canada",
"FoodTypeUnit": "each",
"FoodTypePrice": 0.75
},
{
"FoodTypeName": "MA Little Neck Clam",
"FoodTypeOrigin": "Nauset Inlet, MA, USA",
"FoodTypeUnit": "each",
"FoodTypePrice": 0.56
},
{
"FoodTypeName": "Peconic Bay Scallop",
"FoodTypeOrigin": "Peconic Bay, NY, USA",
"FoodTypeUnit": "pound",
"FoodTypePrice": 19.95
}
]
},
{
"FoodName": "mushrooms",
"FoodLastUpdated": "2021-01-31T00:00:00",
"FoodTypeOptionsList": [
{
"FoodTypeName": "Italian Porcini",
"FoodTypeOrigin": "Puglia, IT",
"FoodTypeUnit": "gram",
"FoodTypePrice": 11.22
},
{
"FoodTypeName": "Polish Pfefferlinge",
"FoodTypeOrigin": "Wroclaw, PL",
"FoodTypeUnit": "gram",
"FoodTypePrice": 7.56
},
{
"FoodTypeName": "Washington State Morel",
"FoodTypeOrigin": "Seattle, WA, USA",
"FoodTypeUnit": "ounce",
"FoodTypePrice": 22.95
}
]
}
]
JSON file length: 1,237 lines: 52 Ln: 52 Col: 2 Pos: 1,238 Windows (CR LF) UTF-8 INS

Using the CSE and .Net to obtain the contents of a web page

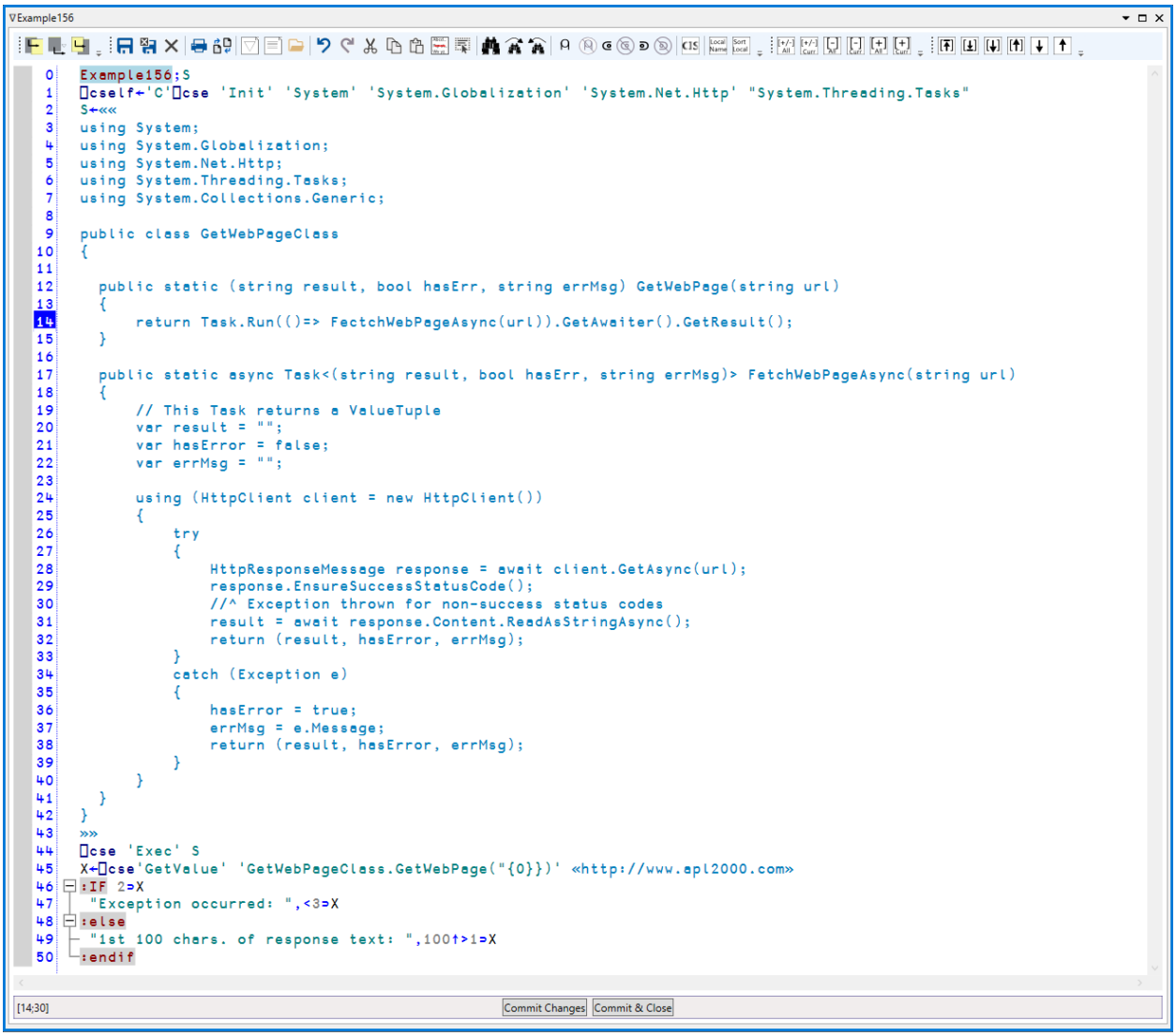
Example #156:

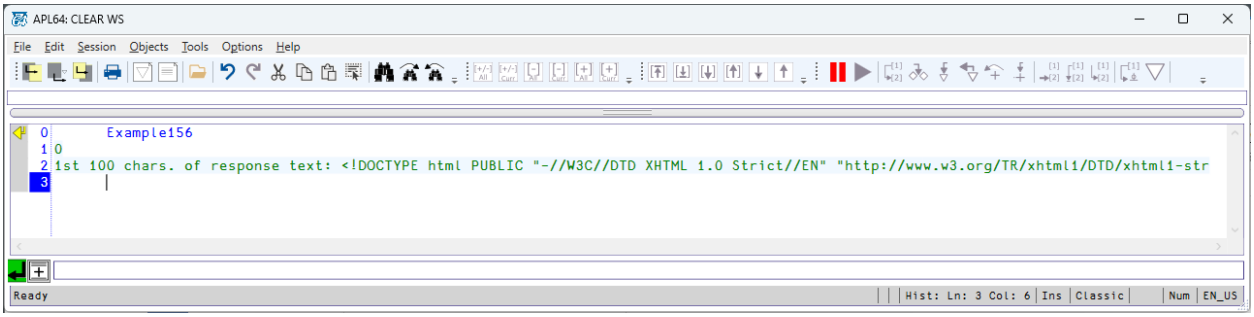
In this example:

- .Net HttpClient will be used to access a public web page and capture its content in APL64
- Using a .Net asynchronous Task in the CSE
- Passing the APL64 string URL as the url argument to the GetWebPage() method
- Using the CSE GetValue action to obtain a .Net ValueTuple as an APL64 vector of values. APL64 cannot set the values of a ValueTuple from an APL vector, because APL64 converts an APL vector to a .Net vector.

```
Example156;S
□cself←'C'□cse 'Init' 'System' 'System.Globalization' 'System.Net.Http' "System.Threading.Tasks"
S←««
using System;
using System.Globalization;
using System.Net.Http;
using System.Threading.Tasks;
using System.Collections.Generic;

public class GetWebPageClass
{

 public static (string result, bool hasErr, string errMsg) GetWebPage(string url)
 {
   return Task.Run(()=> FetchWebPageAsync(url)).GetAwaiter().GetResult();
 }

 public static async Task<(string result, bool hasErr, string errMsg)> FetchWebPageAsync(string url)
 {
   // This Task returns a ValueTuple
   var result = "";
   var hasError = false;
   var errMsg = "";

   using (HttpClient client = new HttpClient())
   {
     try
     {
       HttpResponseMessage response = await client.GetAsync(url);
```

```
            response.EnsureSuccessStatusCode();
            //^ Exception thrown for non-success status codes
            result = await response.Content.ReadAsStringAsync();
            return (result, hasError, errMsg);
        }
        catch (Exception e)
        {
            hasError = true;
            errMsg = e.Message;
            return (result, hasError, errMsg);
        }
    }
  }
}
»»
⎕cse 'Exec' S
X←⎕cse'GetValue' 'GetWebPageClass.GetWebPage("{0}})' «http://www.apl2000.com»
:IF 2⊃X
 "Exception occurred: ",<3⊃X
:else
 "1st 100 chars. of response text: ",100↑>1⊃X
:endif
```





Using the CSE to parse XML using Linq

Example #157:

In this example an XML document will be created, loaded and the values of specific elements of that document queried and modified using the Linq to Xml query features of .Net.

The APL64 ‘Example157’ function using Linq expressions:

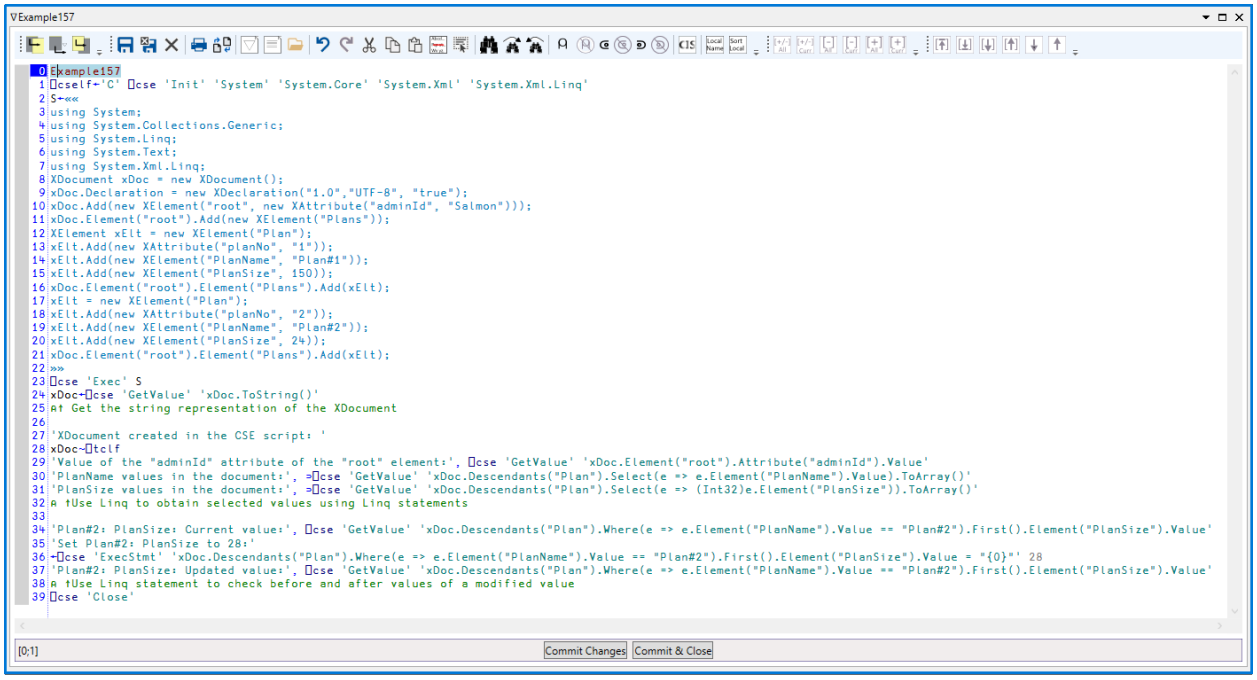
- Obtains selected values in the XDocument
- Updates a selected value in the XDocument
- Saves the XDocument to a physical Xml-format file
- Loads the file into an XDocument

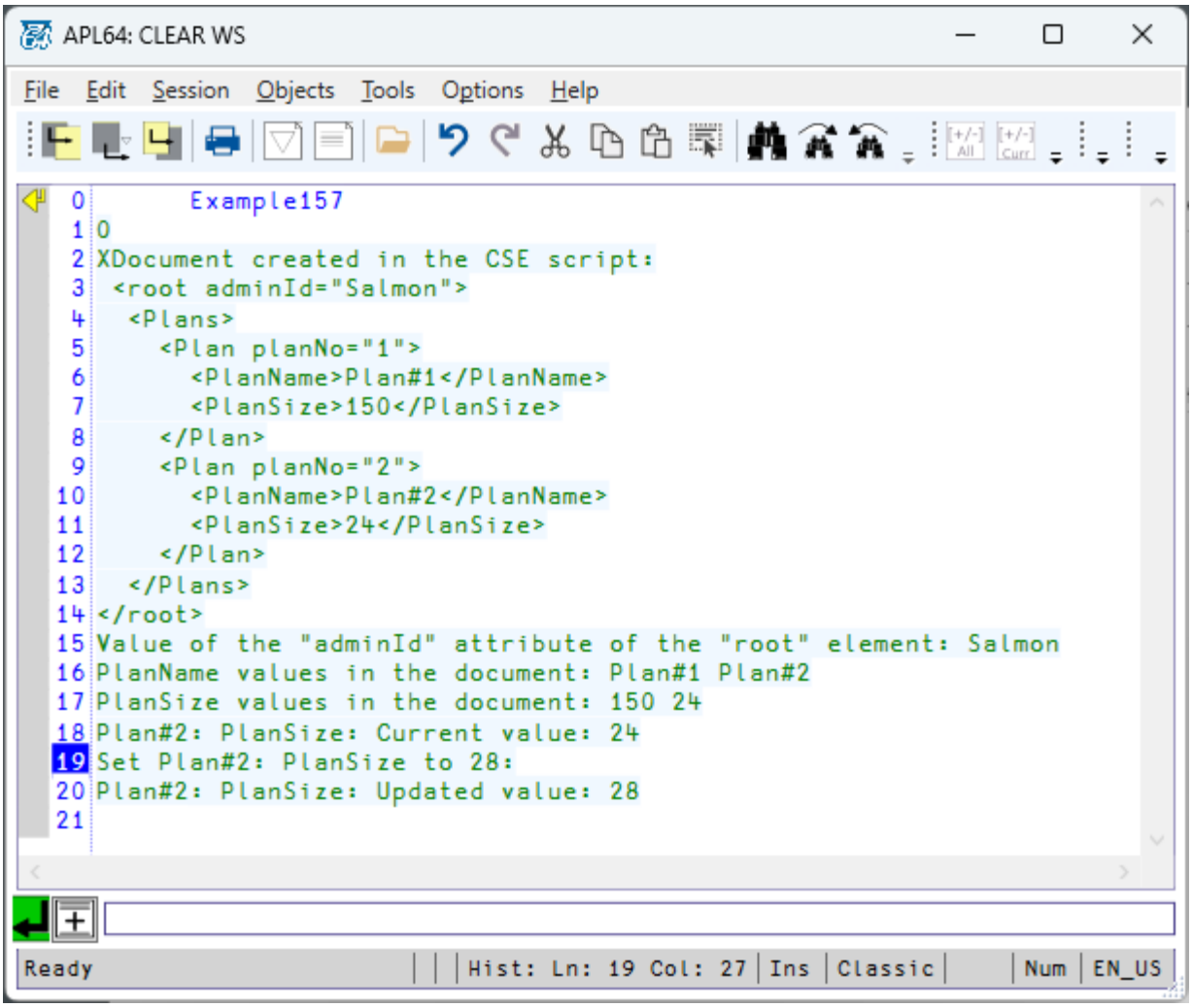
Check out the quick summary of Linq (Language Integrated Query) in the Microsoft .Net. Complete documentation of Linq is available on MSDN.

The APL64 □XML system function is a convenient alternative to using the CSE to parse XML-format documents.

```
Example157
□cself←'C' □cse 'Init' 'System' 'System.Core' 'System.Xml' 'System.Xml.Linq'
S←««
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Linq;
XDocument xDoc = new XDocument();
xDoc.Declaration = new XDeclaration("1.0","UTF-8", "true");
xDoc.Add(new XElement("root", new XAttribute("adminId", "Salmon")));
xDoc.Element("root").Add(new XElement("Plans"));
XElement xElt = new XElement("Plan");
xElt.Add(new XAttribute("planNo", "1"));
xElt.Add(new XElement("PlanName", "Plan#1"));
xElt.Add(new XElement("PlanSize", 150));
xDoc.Element("root").Element("Plans").Add(xElt);
xElt = new XElement("Plan");
xElt.Add(new XAttribute("planNo", "2"));
xElt.Add(new XElement("PlanName", "Plan#2"));
xElt.Add(new XElement("PlanSize", 24));
xDoc.Element("root").Element("Plans").Add(xElt);
»»
□cse 'Exec' S
xDoc←□cse 'GetValue' 'xDoc.ToString()'
⍝↑ Get the string representation of the XDocument

'XDocument created in the CSE script: '
xDoc~□tclf
'Value of the "adminId" attribute of the "root" element:', □cse 'GetValue'
'xDoc.Element("root").Attribute("adminId").Value'
```

```
'PlanName values in the document:', ⊃□cse 'GetValue' 'xDoc.Descendants("Plan").Select(e => e.Element("PlanName").Value).ToArray()'
'PlanSize values in the document:', ⊃□cse 'GetValue' 'xDoc.Descendants("Plan").Select(e => (Int32)e.Element("PlanSize")).ToArray()'
⍝ ↑Use Linq to obtain selected values using Linq statements

'Plan#2: PlanSize: Current value:', □cse 'GetValue' 'xDoc.Descendants("Plan").Where(e => e.Element("PlanName").Value == "Plan#2").First().Element("PlanSize").Value'
'Set Plan#2: PlanSize to 28:'
←□cse 'ExecStmt' 'xDoc.Descendants("Plan").Where(e => e.Element("PlanName").Value == "Plan#2").First().Element("PlanSize").Value = "{0}"' 28
'Plan#2: PlanSize: Updated value:', □cse 'GetValue' 'xDoc.Descendants("Plan").Where(e => e.Element("PlanName").Value == "Plan#2").First().Element("PlanSize").Value'
⍝ ↑Use Linq statement to check before and after values of a modified value
□cse 'Close'
```





Details of the C# Event Args Object

Example #162:

When a C# object event has been subscribed by APL64 via the CSE 'AddEventHandler' method and that C# event fires, the APL64 application-specific event handler function is executed. That event handler function receives information, via its required right argument, about the event which fired. The information in that right argument can be used by the APL64 event handler function to process that event using both the state of the APL64 environment and the state of the CSE instance which contains the C# object which fired the event.

The 5th element of that right argument is an array whose 1st column provides the event argument property names. The .Net event argument properties will vary depending on the .Net event which is subscribed by APL64. If an event argument property value has a .Net data type which has an analogous APL64 data type, the 2nd column of that array contains that property value. Using this array, the value of many event argument property values can be easily accessed by the APL64 event handler function.

Sometimes the event argument property has a value with a .Net data type that has no analogous APL64 data type. In this case the value in the 2nd column of the array is the message 'See _cseEvt for this object'. If it is necessary for the APL64 event handler function to get or set the value of such an event argument, the 4th element of the right argument of the APL64 event handler function can be used.

The 4th element of the right argument provided by the CSE to the APL64 event handler function is the key in the .Net dictionary (_cseEvt) of the event argument property object. Using CSE methods it is possible determine the object model of such an event argument property object.

In example #162 the event arguments returned to the APL64 programmer-defined event handler function when 'MyEvent' is fired as a result of using the Class1.MyMethod() method, include a .Net DateTime value which does not have an analogous representation in APL64. With this knowledge the DateTime value can be coerced in .Net to a text vector, e.g. 'mm/dd/ccyy', and consumed by APL64.

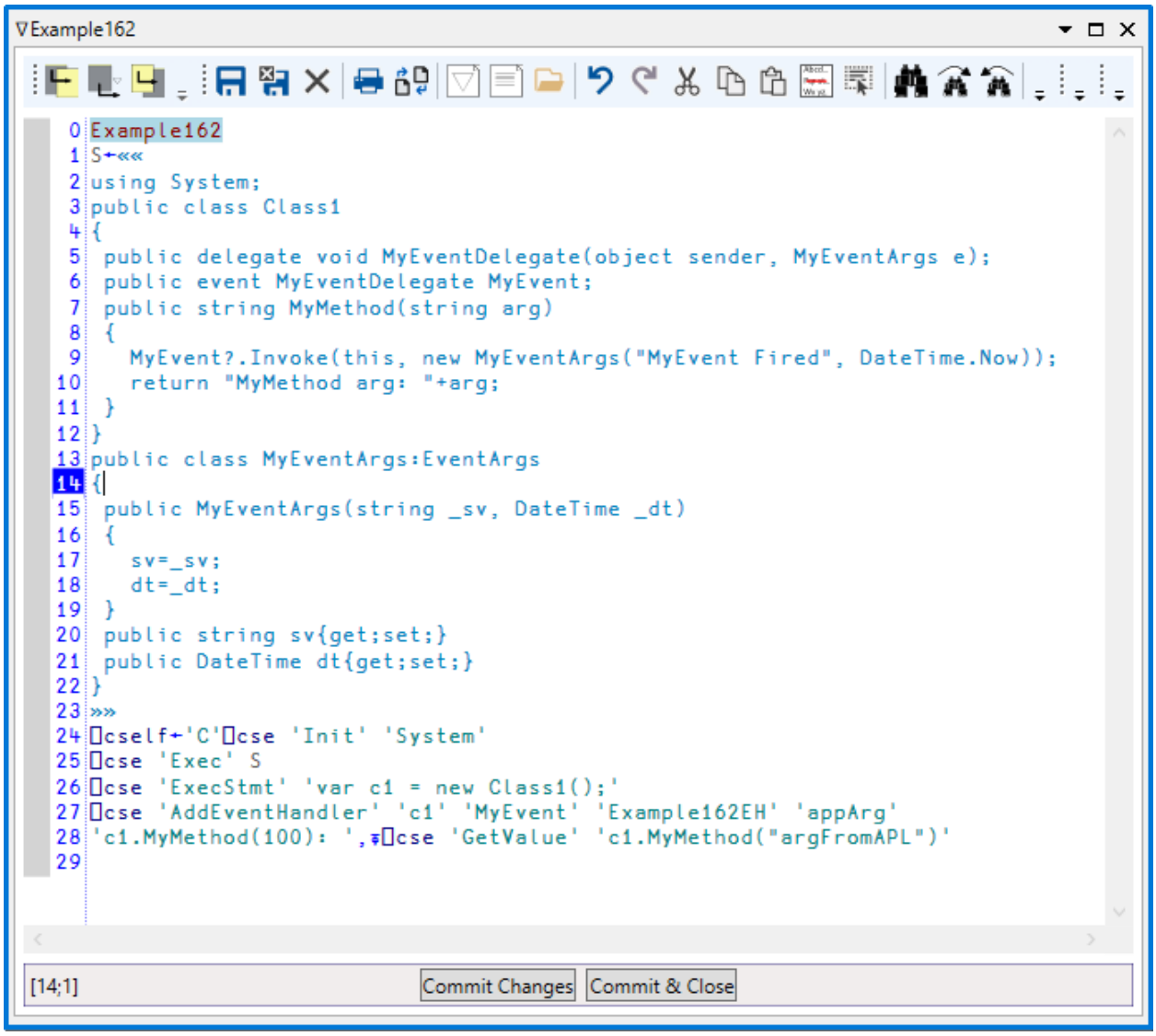
APL64 EventArgsDetails function is called several times with different arguments in the Example162EH event handler function to illustrate how to use the CSE-provided information.

Be sure to carefully examine the APL64 EventArgsDetails function that illustrates techniques for working with the C# _cseEvt dictionary which contains the C# eventargs object.

Create the Example162 function using this definition:

```
Example162
S←««
using System;
public class Class1
{
 public delegate void MyEventDelegate(object sender, MyEventArgs e);
 public event MyEventDelegate MyEvent;
 public string MyMethod(string arg)
 {
  MyEvent?.Invoke(this, new MyEventArgs("MyEvent Fired", DateTime.Now));
  return "MyMethod arg: "+arg;
 }
}
public class MyEventArgs:EventArgs
{
 public MyEventArgs(string _sv, DateTime _dt)
 {
  sv=_sv;
  dt=_dt;
 }
 public string sv{get;set;}
```

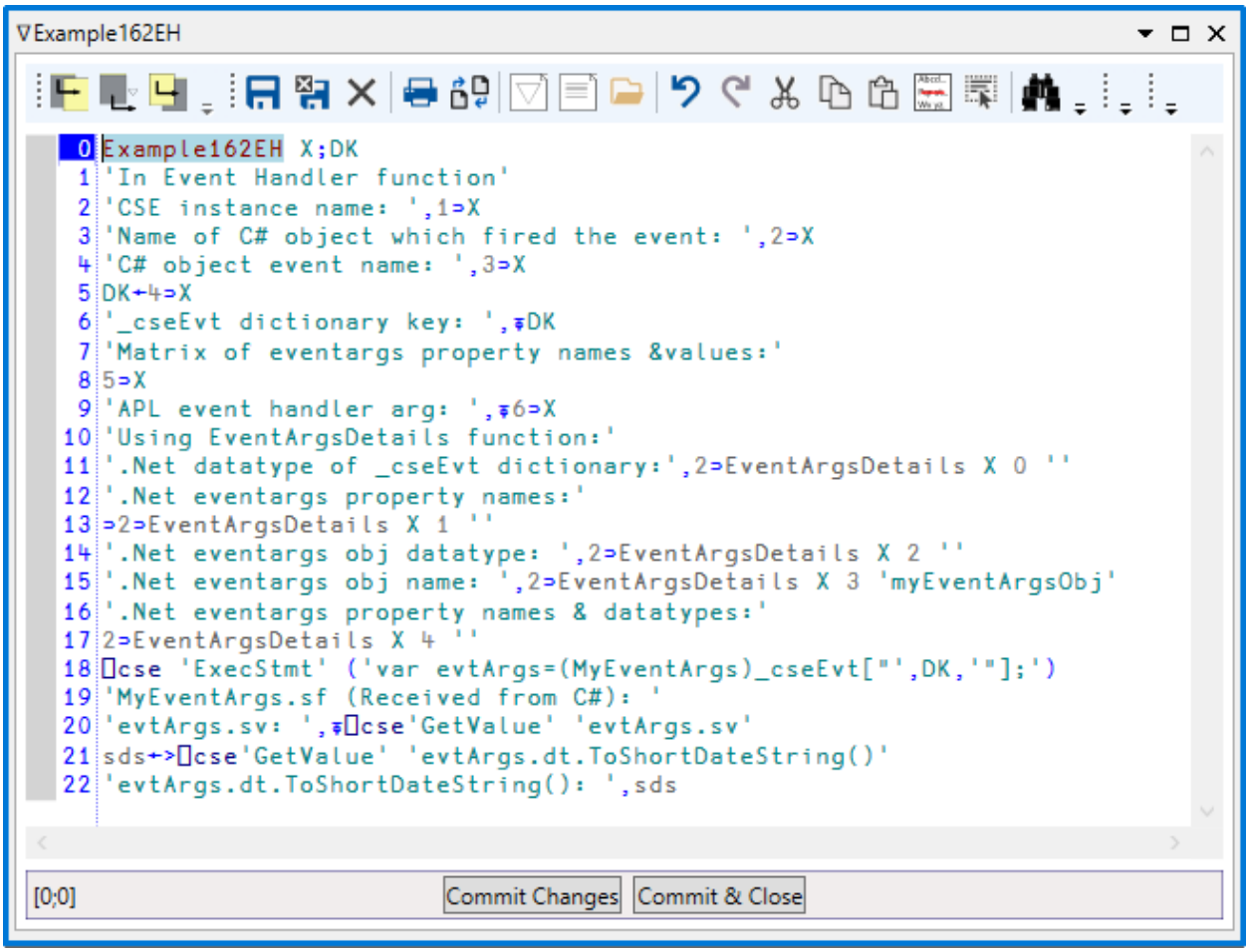
```
 public DateTime dt{get;set;}
}
»»
□cself←'C'□cse 'Init' 'System'
□cse 'Exec' S
□cse 'ExecStmt' 'var c1 = new Class1();'
□cse 'AddEventHandler' 'c1' 'MyEvent' 'Example162EH' 'appArg'
'c1.MyMethod(100): ',⍕□cse 'GetValue' 'c1.MyMethod("argFromAPL")'
```



Create the Example162EH function using this definition

```
Example162EH X;DK
'In Event Handler function'
'CSE instance name: ',1⊃X
'Name of C# object which fired the event: ',2⊃X
```

```
'C# object event name: ',3⊃X
DK←4⊃X
'_cseEvt dictionary key: ', ⍕ DK
'Matrix of eventargs property names &values:'
5⊃X
'APL event handler arg: ', ⍕ 6⊃X
'Using EventArgsDetails function:'
'.Net datatype of _cseEvt dictionary:',2⊃EventArgsDetails X 0 ''
'.Net eventargs property names:'
⊃2⊃EventArgsDetails X 1 ''
'.Net eventargs obj datatype: ',2⊃EventArgsDetails X 2 ''
'.Net eventargs obj name: ',2⊃EventArgsDetails X 3 'myEventArgsObj'
'.Net eventargs property names & datatypes:'
2⊃EventArgsDetails X 4 ''
□cse 'ExecStmt' ('var evtArgs=(MyEventArgs)_cseEvt["',DK,'"];')
'MyEventArgs.sf (Received from C#): '
'evtArgs.sv: ', ⍕ □cse'GetValue' 'evtArgs.sv'
sds←>□cse'GetValue' 'evtArgs.dt.ToShortDateString()'
'evtArgs.dt.ToShortDateString(): ',sds
□DEF F
```

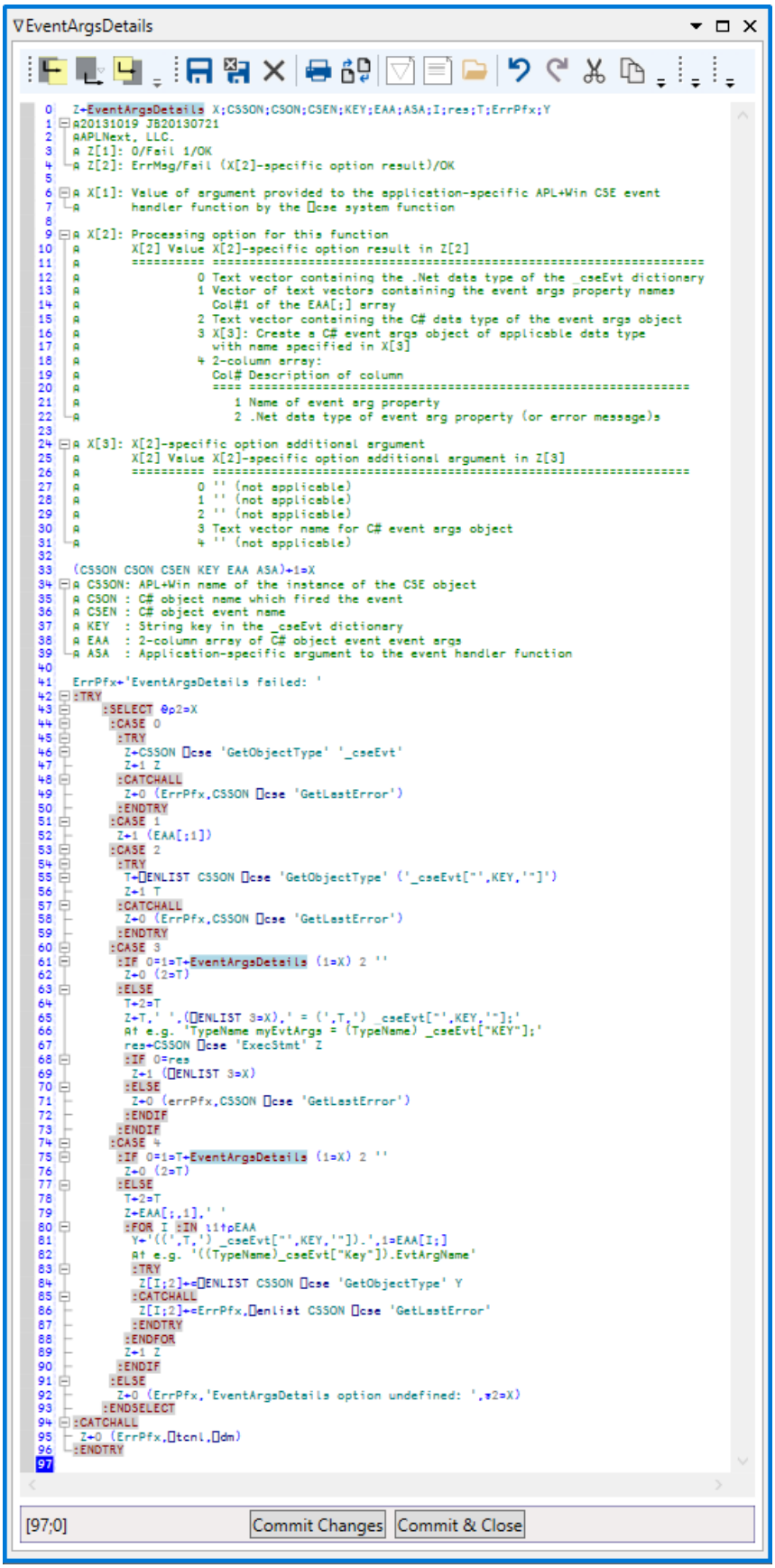


Create the EventArgsDetails function using this definition:

```
Z←EventArgsDetails X;CSSON;CSON;CSEN;KEY;EAA;ASA;I;res;T;ErrPfx;Y
⍝20131019 JB20130721
⍝APLNext, LLC.
⍝ Z[1]: 0/Fail 1/OK
⍝ Z[2]: ErrMsg/Fail (X[2]-specific option result)/OK

⍝ X[1]: Value of argument provided to the application-specific APL+Win CSE event
⍝       handler function by the □cse system function

⍝ X[2]: Processing option for this function
⍝       X[2] Value X[2]-specific option result in Z[2]
⍝       ========== ========================================================================
⍝             0 Text vector containing the .Net data type of the _cseEvt dictionary
⍝             1 Vector of text vectors containing the event args property names
⍝               Col#1 of the EAA[;] array
⍝             2 Text vector containing the C# data type of the event args object
```

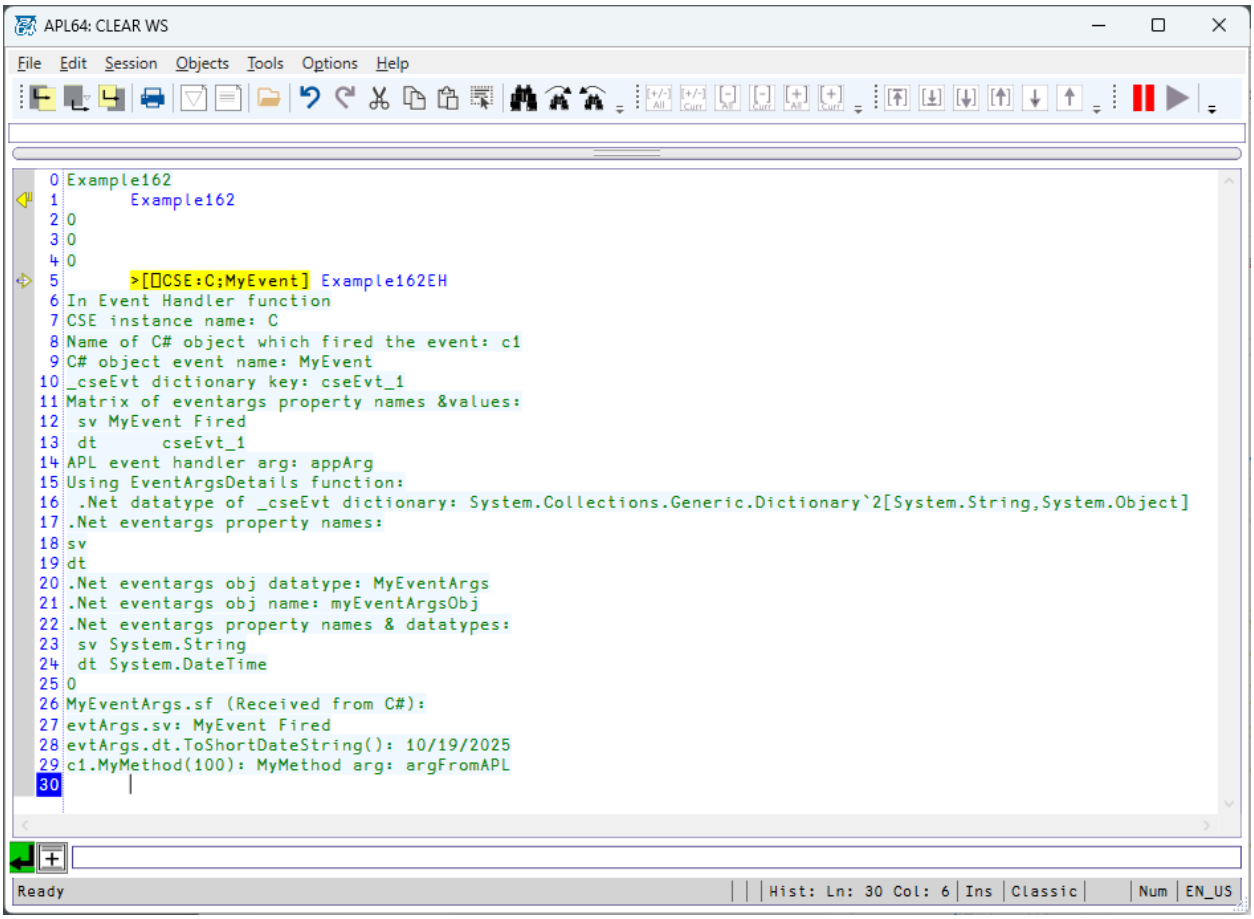
```
⍝             3 X[3]: Create a C# event args object of applicable data type
⍝               with name specified in X[3]
⍝             4 2-column array:
⍝               Col# Description of column
⍝               ==== ============================================================
⍝                  1 Name of event arg property
⍝                  2 .Net data type of event arg property (or error message)s

⍝ X[3]: X[2]-specific option additional argument
⍝     X[2] Value X[2]-specific option additional argument in Z[3]
⍝     ========== ======================================================================
⍝             0 '' (not applicable)
⍝             1 '' (not applicable)
⍝             2 '' (not applicable)
⍝             3 Text vector name for C# event args object
⍝             4 '' (not applicable)

(CSSON CSON CSEN KEY EAA ASA)←1⊃X
⍝ CSSON: APL+Win name of the instance of the CSE object
⍝ CSON : C# object name which fired the event
⍝ CSEN : C# object event name
⍝ KEY  : String key in the _cseEvt dictionary
⍝ EAA  : 2-column array of C# object event event args
⍝ ASA  : Application-specific argument to the event handler function

ErrPfx←'EventArgsDetails failed: '
:TRY
   :SELECT ⍬⍴2⊃X
    :CASE 0
     :TRY
      Z←CSSON □cse 'GetObjectType' '_cseEvt'
      Z←1 Z
     :CATCHALL
      Z←0 (ErrPfx,CSSON □cse 'GetLastError')
     :ENDTRY
    :CASE 1
     Z←1 (EAA[;1])
    :CASE 2
     :TRY
      T←□ENLIST CSSON □cse 'GetObjectType' ('_cseEvt["',KEY,'"]')
      Z←1 T
     :CATCHALL
      Z←0 (ErrPfx,CSSON □cse 'GetLastError')
```

```
      :ENDTRY
     :CASE 3
      :IF 0=1⊃T←EventArgsDetails (1⊃X) 2 ''
       Z←0 (2⊃T)
      :ELSE
       T←2⊃T
       Z←T,' ',(⎕ENLIST 3⊃X),' = (',T,') _cseEvt["'",KEY,'"];'
       ⍝↑ e.g. 'TypeName myEvtArgs = (TypeName) _cseEvt["KEY"];'
       res←CSSON ⎕cse 'ExecStmt' Z
       :IF 0=res
        Z←1 (⎕ENLIST 3⊃X)
       :ELSE
        Z←0 (errPfx,CSSON ⎕cse 'GetLastError')
       :ENDIF
      :ENDIF
     :CASE 4
      :IF 0=1⊃T←EventArgsDetails (1⊃X) 2 ''
       Z←0 (2⊃T)
      :ELSE
       T←2⊃T
       Z←EAA[;,1],' '
       :FOR I :IN ⍳1↑⍴EAA
        Y←'((',T,') _cseEvt["'",KEY,'"]).',1⊃EAA[I;]
        ⍝↑ e.g. '((TypeName)_cseEvt["Key"]).EvtArgName'
        :TRY
         Z[I;2]←⊂⎕ENLIST CSSON ⎕cse 'GetObjectType' Y
        :CATCHALL
         Z[I;2]←⊂ErrPfx,⎕enlist CSSON ⎕cse 'GetLastError'
        :ENDTRY
       :ENDFOR
       Z←1 Z
      :ENDIF
     :ELSE
      Z←0 (ErrPfx,'EventArgsDetails option undefined: ',⍕2⊃X)
    :ENDSELECT
:CATCHALL
 Z←0 (ErrPfx,⎕tcnl,⎕dm)
:ENDTRY
```


∇EventArgsDetails
Z←EventArgsDetails X;CSSON;CSON;CSEN;KEY;EAA;ASA;I;res;T;ErrPfx;Y
⍝20131019 JB20130721
⍝APLNext, LLC.
⍝ Z[1]: 0/Fail 1/OK
⍝ Z[2]: ErrMsg/Fail (X[2]-specific option result)/OK
⍝ X[1]: Value of argument provided to the application-specific APL+Win CSE event
⍝ handler function by the ⎕cse system function
⍝ X[2]: Processing option for this function
⍝ X[2] Value X[2]-specific option result in Z[2]
⍝ 0 Text vector containing the .Net data type of the _cseEvt dictionary
⍝ 1 Vector of text vectors containing the event args property names
⍝ Col#1 of the EAA[;] array
⍝ 2 Text vector containing the C# data type of the event args object
⍝ 3 X[3]: Create a C# event args object of applicable data type
⍝ with name specified in X[3]
⍝ 4 2-column array:
⍝ Col# Description of column
⍝ 1 Name of event arg property
⍝ 2 .Net data type of event arg property (or error message)s
⍝ X[3]: X[2]-specific option additional argument
⍝ X[2] Value X[2]-specific option additional argument in Z[3]
⍝ 0 '' (not applicable)
⍝ 1 '' (not applicable)
⍝ 2 '' (not applicable)
⍝ 3 Text vector name for C# event args object
⍝ 4 '' (not applicable)
(CSSON CSON CSEN KEY EAA ASA)←1↓X
⍝ CSSON: APL+Win name of the instance of the CSE object
⍝ CSON : C# object name which fired the event
⍝ CSEN : C# object event name
⍝ KEY : String key in the _cseEvt dictionary
⍝ EAA : 2-column array of C# object event event args
⍝ ASA : Application-specific argument to the event handler function
ErrPfx←'EventArgsDetails failed: '
:TRY
:SELECT ⍬⍴2↑X
:CASE 0
:TRY
Z←CSSON ⎕cse 'GetObjectType' '_cseEvt'
Z←1 Z
:CATCHALL
Z←0 (ErrPfx,CSSON ⎕cse 'GetLastError')
:ENDTRY
:CASE 1
Z←1 (EAA[;1])
:CASE 2
:TRY
T←⎕ENLIST CSSON ⎕cse 'GetObjectType' ('_cseEvt["',KEY,'"]')
Z←1 T
:CATCHALL
Z←0 (ErrPfx,CSSON ⎕cse 'GetLastError')
:ENDTRY
:CASE 3
:IF 0=1↑T←EventArgsDetails (1↓X) 2 ''
Z←0 (2↓T)
:ELSE
T←2↓T
Z←T,' ',(⎕ENLIST 3↓X),' = (',T,') _cseEvt["',KEY,'"];'
⍝ At e.g. 'TypeName myEvtArgs = (TypeName) _cseEvt["KEY"];'
res←CSSON ⎕cse 'ExecStmt' Z
:IF 0=res
Z←1 (⎕ENLIST 3↓X)
:ELSE
Z←0 (errPfx,CSSON ⎕cse 'GetLastError')
:ENDIF
:ENDIF
:CASE 4
:IF 0=1↑T←EventArgsDetails (1↓X) 2 ''
Z←0 (2↓T)
:ELSE
T←2↓T
Z←EAA[;1],' '
:FOR I :IN ⍳1↑⍴EAA
Y←'((',T,') _cseEvt["',KEY,'"]).',1↓EAA[I;]
⍝ At e.g. '((TypeName)_cseEvt["Key"]).EvtArgName'
:TRY
Z[I;2]←⎕ENLIST CSSON ⎕cse 'GetObjectType' Y
:CATCHALL
Z[I;2]←⊂ErrPfx,⎕enlist CSSON ⎕cse 'GetLastError'
:ENDTRY
:ENDFOR
Z←1 Z
:ENDIF
:ELSE
Z←0 (ErrPfx,'EventArgsDetails option undefined: ',⍕2↓X)
:ENDSELECT
:CATCHALL
Z←0 (ErrPfx,⎕tcnl,⎕dm)
:ENDTRY
[97;0]
Commit Changes
Commit & Close

When the Example162 function is run:

- The CSE script is executed creating the Class1 class containing MyMethod() and MyEvent
- The Example162EH function is subscribed to MyEvent
- Class1.MyMethod(100) is run which causes MyEvent to fire
- The Example162EH function handles MyEvent and displays event-level information
- The EventArgDetails function is run within the Example162EH function and displays additional event-level information



Returning Information Back to C# from an APL64 Event Handler

Example #170

Some inheritors of the .Net EventArgs class include properties which can be set in the APL64 event handler function so that they can be processed by the C# method which fired the .Net vents. Examples of this include keyboard events like 'KeyUp' with the 'Handled' property and the Closing event with the 'Cancel' property.

If it is necessary to return information back from the APL64 event handler function to the C# method which fired the event subscribed by APL64, a .Net custom event can be defined. The basics of this custom event scenario have been previously described in this document.

In this example the APL64 event handler function returns the value of a string back to the C# method which fired the custom event.

For simplicity, this example illustrates one property in the custom event args class definition, but any number of properties can be defined in such a class. Do not confuse the possibility of multiple properties in a .Net custom event args class with the CSE (actually .Net routed event) requirement that an APL64 event handler function must have exactly one argument and no result.

The CSE script for this example defines two classes, 'Class' containing the C# method 'MyMethod' which performs the C# application system processing role in this example and the 'MyEvent' .Net custom event. This CSE script also defines the .Net custom event args class 'MyEventArgs' which contains the 'string sv' property. The value of this property will be received by the APL64 event handler function when the 'MyEvent' fires.

Notice that the C# 'MyMethod' method sends the message "From Calculator" as the value of the 'MyEventArgs.sv' property value to the APL64 event handler function which has subscribed to the 'MyEvent' event.

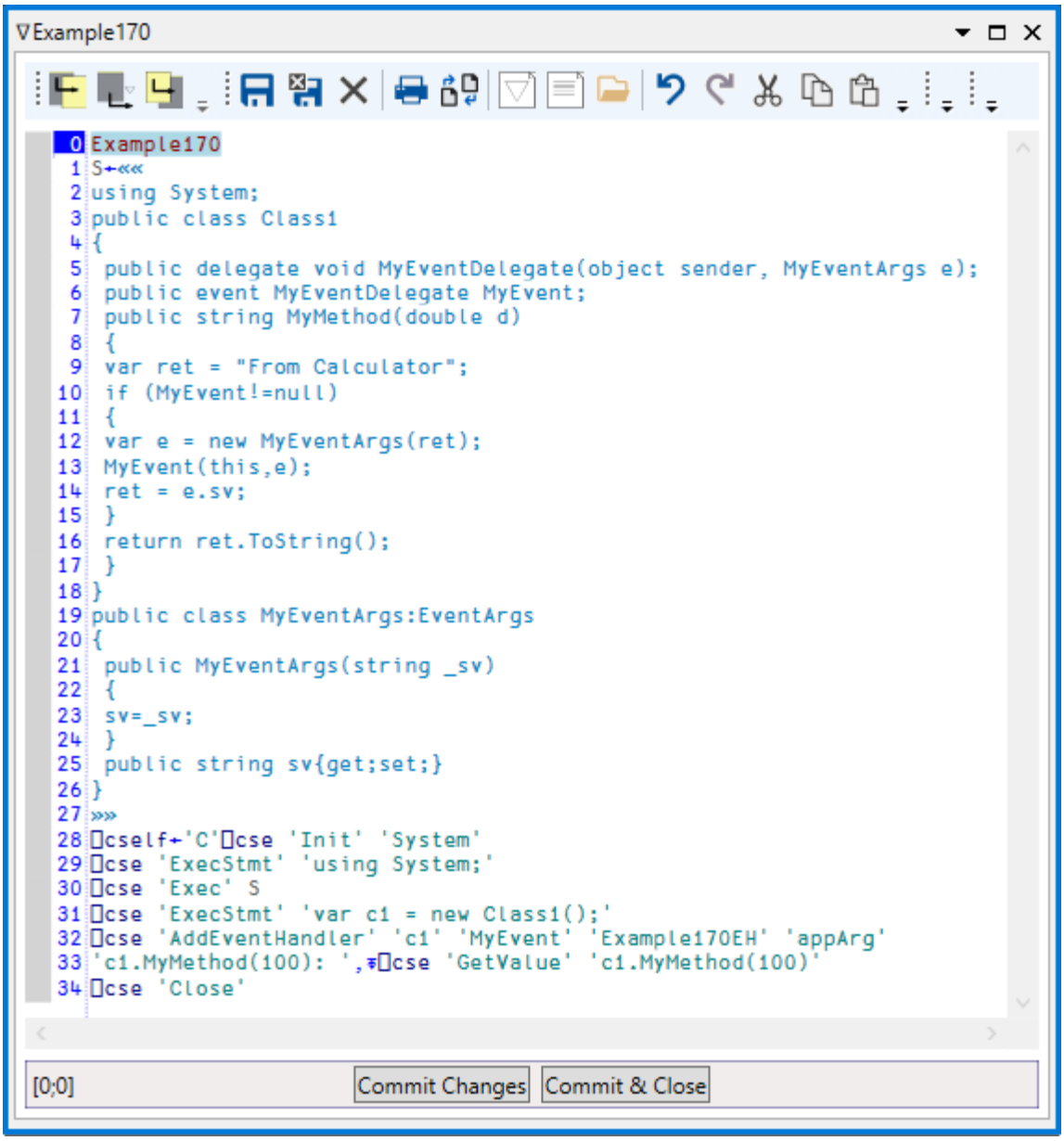
After the C# 'MyMethod' method fires the 'MyEvent' event, the processing flow is interrupted until the execution of the 'Example170EH' APL64 event handler function is complete. That APL64 event handler function send the message "response from APL sent via MyEventsArgs.sv' back to the C# method.

For purposes of this example, the C# method returns the value of the 'MyEventArgs.sv' property as the result of the method so that it can be obtained by the 'Example170' APL64 function which called the 'Class1.MyMethod' method. In a production environment this behavior of the C# method is not required.

Create the Example170 function using this definition:

```
Example170
S←««
using System;
public class Class1
{
 public delegate void MyEventDelegate(object sender, MyEventArgs e);
 public event MyEventDelegate MyEvent;
 public string MyMethod(double d)
 {
 var ret = "From Calculator";
 if (MyEvent!=null)
 {
 var e = new MyEventArgs(ret);
 MyEvent(this,e);
 ret = e.sv;
```

```
 }
 return ret.ToString();
 }
}
public class MyEventArgs:EventArgs
{
 public MyEventArgs(string _sv)
 {
 sv=_sv;
 }
 public string sv{get;set;}
}
»»
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System;'
⎕cse 'Exec' S
⎕cse 'ExecStmt' 'var c1 = new Class1();'
⎕cse 'AddEventHandler' 'c1' 'MyEvent' 'Example170EH' 'appArg'
'c1.MyMethod(100): ', ⍕ ⎕cse 'GetValue' 'c1.MyMethod(100)'
⎕cse 'Close'
```

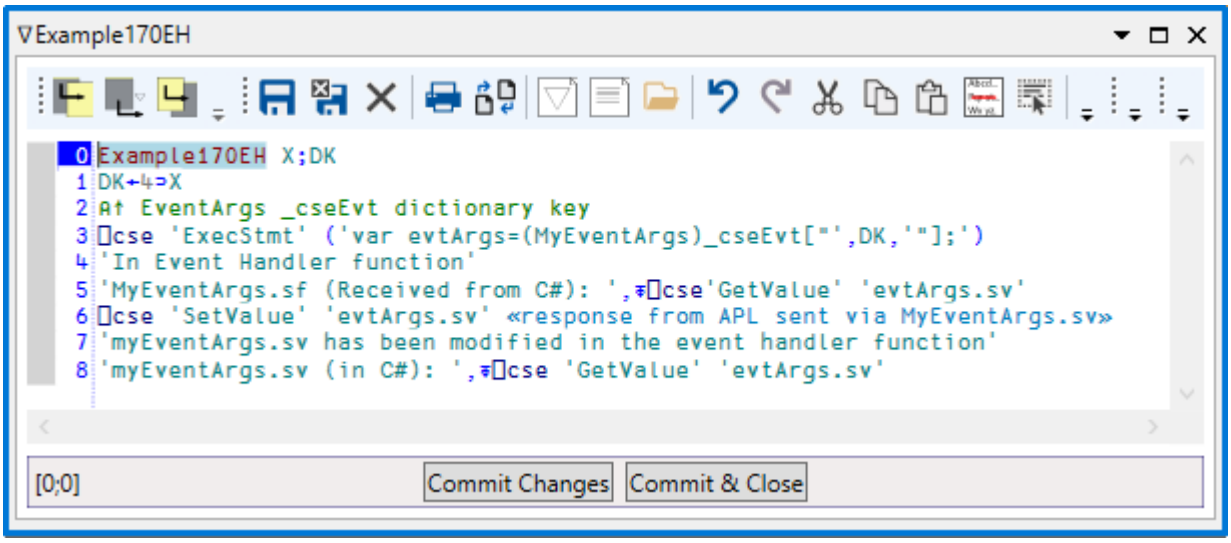


Create the Example17EH function using this definition:

```
Example170EH X;DK
DK←4⊃X
⍝↑ EventArgs _cseEvt dictionary key
cse 'ExecStmt' ('var evtArgs=(MyEventArgs)_cseEvt["',DK,'"];')
'In Event Handler function'
'MyEventArgs.sf (Received from C#): ',⍕cse'GetValue' 'evtArgs.sv'
cse 'SetValue' 'evtArgs.sv' «response from APL sent via MyEventArgs.sv»
'myEventArgs.sv has been modified in the event handler function'
'myEventArgs.sv (in C#): ',⍕cse 'GetValue' 'evtArgs.sv'
```

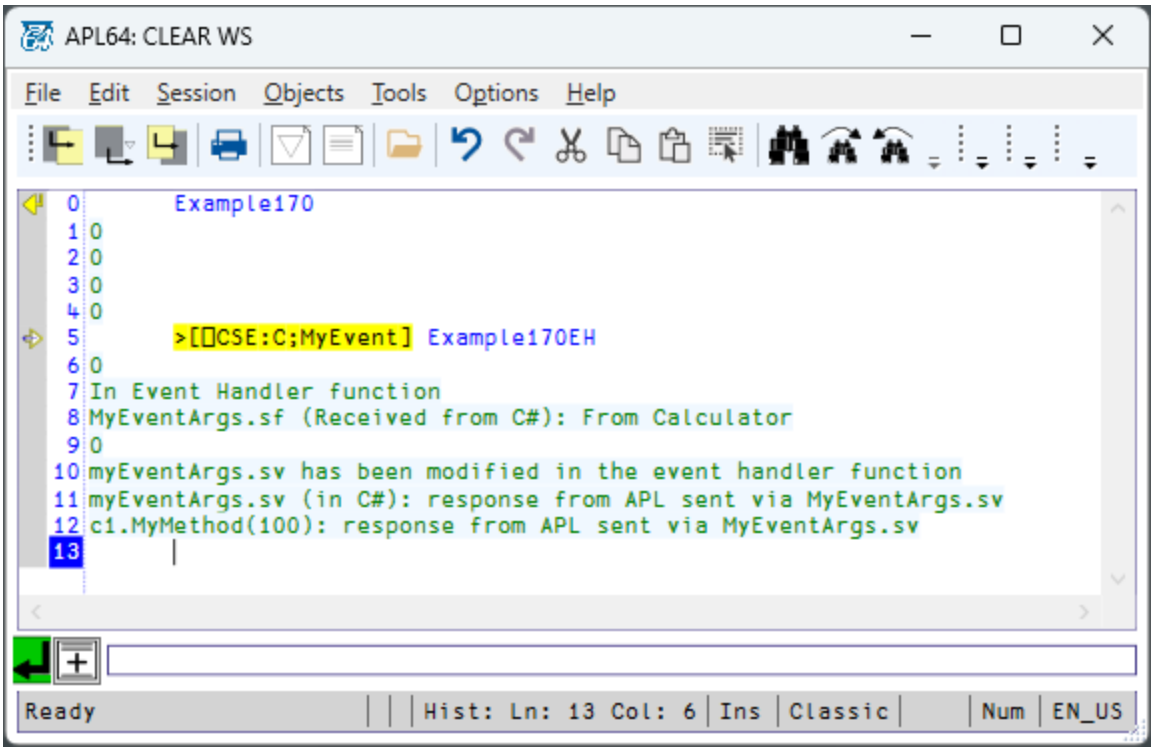
The ‘Example170EH’ APL64 event handler function illustrates how to:

- Define a C# variable to contain the custom event args class instance ‘evtArgs’ using the CSE ‘_cseEvt’ dictionary of event arguments and the dictionary key ‘DK’ available as the 4th element of the argument of the APL64 event handler function.
- Get the value of the ‘evtArgs.sv’ property from C# to APL64
- Modify the value of the ‘evtArgs.sv’ property so that the C# method which fired the event can receive this value.



In this example the APL64 event handler function not only obtains the value of the ‘MyEventArgs.sv’ property from the CSE instance, but it also modifies the value of this property in the CSE instance. The modified value of this property is then received by the C# method which fired the event. That C# method reports that modified value back to the APL64 function which calls the C# method.

When this example is run the value of the ‘MyEventArgs.sv’ property received from C# and modified by the APL64 event handler function is illustrated.



Using the CSE with .Net Cryptography

Example #165:

The .Net includes the well-documented System.Security.Cryptography namespace which provides support for encrypting and decrypting data.

Create the Example165 function from this definition and run the function.

```
Example165;S
S←««
using System;
using System.IO;
using System.Security.Cryptography;
public class AesEncryptionService
{
   public static string AesKeyBase64 = "";
   public static string AesIVBase64 = "";
   //^ Base64 representation of the AES Key is used because APL cannot handle byte[] directly

   public static void NewAESKeyAndIV()
   {
      using (Aes aes = Aes.Create())
      {
         AesKeyBase64 = Convert.ToBase64String(aes.Key, Base64FormattingOptions.None);
```

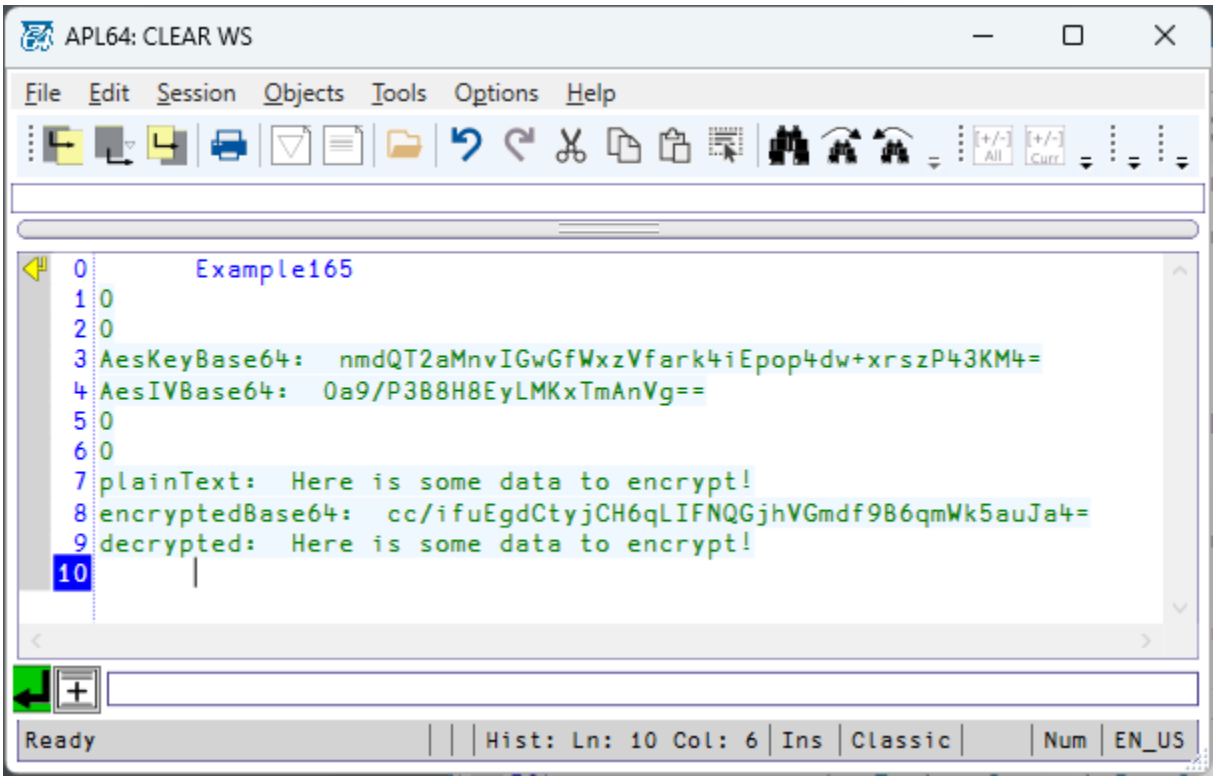
```
        AesIVBase64 = Convert.ToBase64String(aes.IV, Base64FormattingOptions.None);
      }
    }

    public static string EncryptStringToBase64String_Aes(string plainText)
    {
      if (String.IsNullOrWhiteSpace(plainText))
        throw new ArgumentNullException("plainText is null or empty!");
      byte[] encrypted;
      using (Aes aesAlg = Aes.Create())
      {
        aesAlg.Key = Convert.FromBase64String(AesKeyBase64);
        aesAlg.IV = Convert.FromBase64String(AesIVBase64);
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
        using (MemoryStream msEncrypt = new MemoryStream())
        {
          using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
          {
            using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
            {
              swEncrypt.Write(plainText);
            }
            encrypted = msEncrypt.ToArray();
          }
        }
      }
      return Convert.ToBase64String(encrypted, Base64FormattingOptions.None);
    }

    public static string DecryptStringFromBase64String_Aes(string cipherTextBase64)
    {
      if (String.IsNullOrWhiteSpace(cipherTextBase64))
        throw new ArgumentNullException("cipherTextBase64 is null or empty!");
      string plaintext = null;
      var cipherText = Convert.FromBase64String(cipherTextBase64);
      using (Aes aesAlg = Aes.Create())
      {
        aesAlg.Key = Convert.FromBase64String(AesKeyBase64);
        aesAlg.IV = Convert.FromBase64String(AesIVBase64);

        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
```

```
        using (MemoryStream msDecrypt = new MemoryStream(cipherText))
        {
          using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
          {
            using (StreamReader srDecrypt = new StreamReader(csDecrypt))
            {
              plaintext = srDecrypt.ReadToEnd();
            }
          }
        }
      }
      return plaintext;
    }
}
»»
⎕cselfâ†'C'⎕cse 'Init' 'System' 'System.Security'
⎕cse 'Exec' S
⍝^ User input data to encrypt
⎕cse 'ExecStmt' 'AesEncryptionService.NewAESKeyAndIV();'
aesKeyBase64â†⎕cse 'GetValue' 'AesEncryptionService.AesKeyBase64'
aesIVBase64â†⎕cse 'GetValue' 'AesEncryptionService.AesIVBase64'
"AesKeyBase64: ",aesKeyBase64
"AesIVBase64: ",aesIVBase64
⍝^ Obtain a new set of AES Key and IV
⍝  Store these securely for decryption later

⍝ For this example, the Key and IV will be used immediately
⎕cse 'SetValue' 'AesEncryptionService.AesKeyBase64' aesKeyBase64
⎕cse 'SetValue' 'AesEncryptionService.AesIVBase64' aesIVBase64

plainTextâ†«Here is some data to encrypt!»
"plainText: ",plainText
encryptedBase64â†⎕cse 'GetValue' 'AesEncryptionService.EncryptStringToBase64String_Aes("{0}")'
plainText
"encryptedBase64: ",encryptedBase64
decryptedâ†⎕cse 'GetValue' 'AesEncryptionService.DecryptStringFromBase64String_Aes("{0}")'
encryptedBase64
"decrypted: ",decrypted
```

When Example165 is run:

- New Aes encryption Key and IV are created
- They are used to encrypt the plain text into encrypted text

- The encrypted text is decrypted to a value which matches the original plain text
- Standard cryptographic technology is being used by APL64 in this example, which means that encrypted data can be exchanged between APL64 and other environments.



Example 166B Using the CSE to obtain IP Address Information

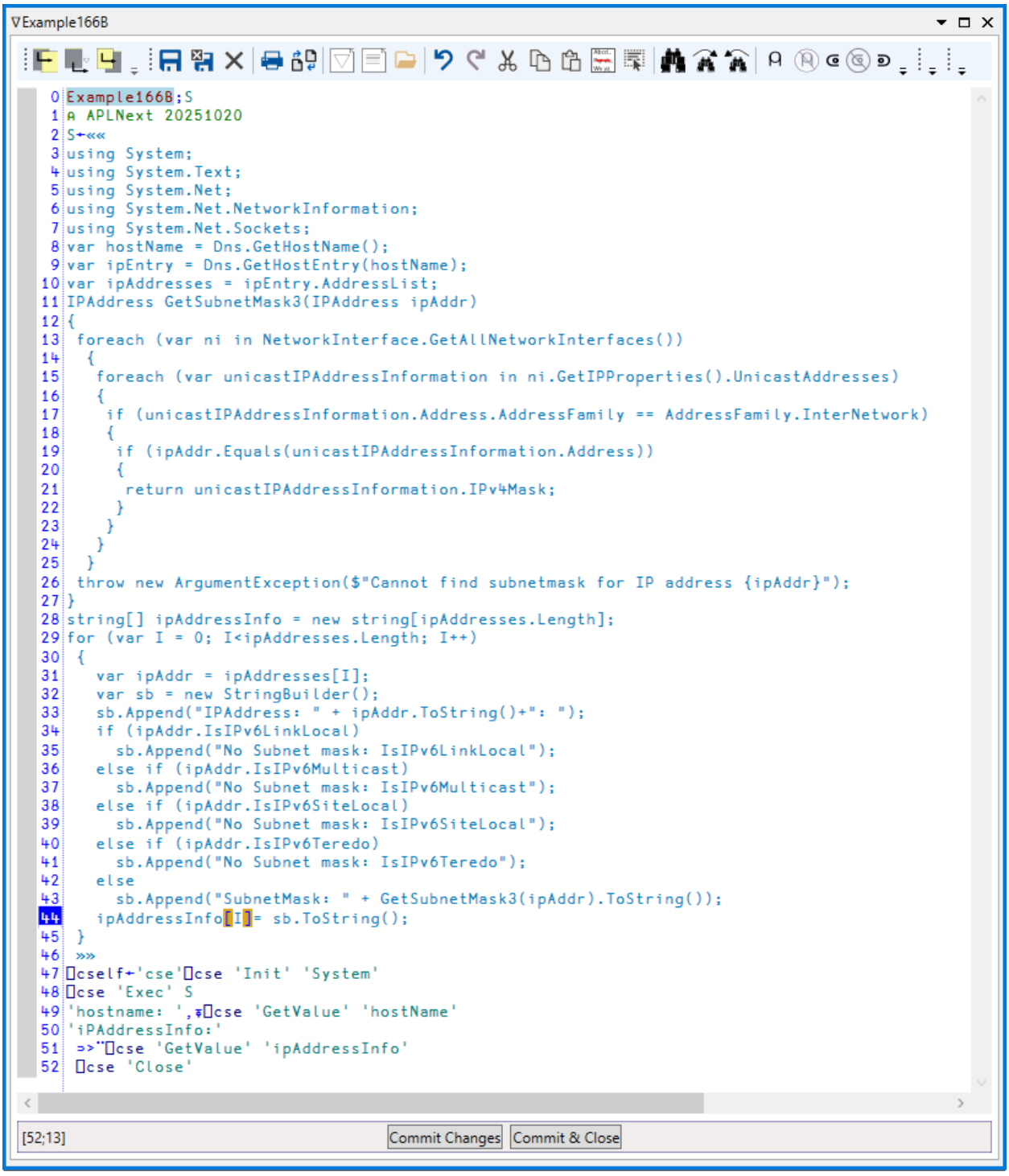
The System.Net namespace can be used to access TCP/IP address information. These examples illustrate obtaining IP Address and Subnet Mask information for the local workstation. Many other capabilities are supported by the System.Net namespace. The CSE script in Example166B defines several .Net objects, created when the script is executed via the CSE 'Exec' method, which APL64 can access via the CSE 'GetValue' method:

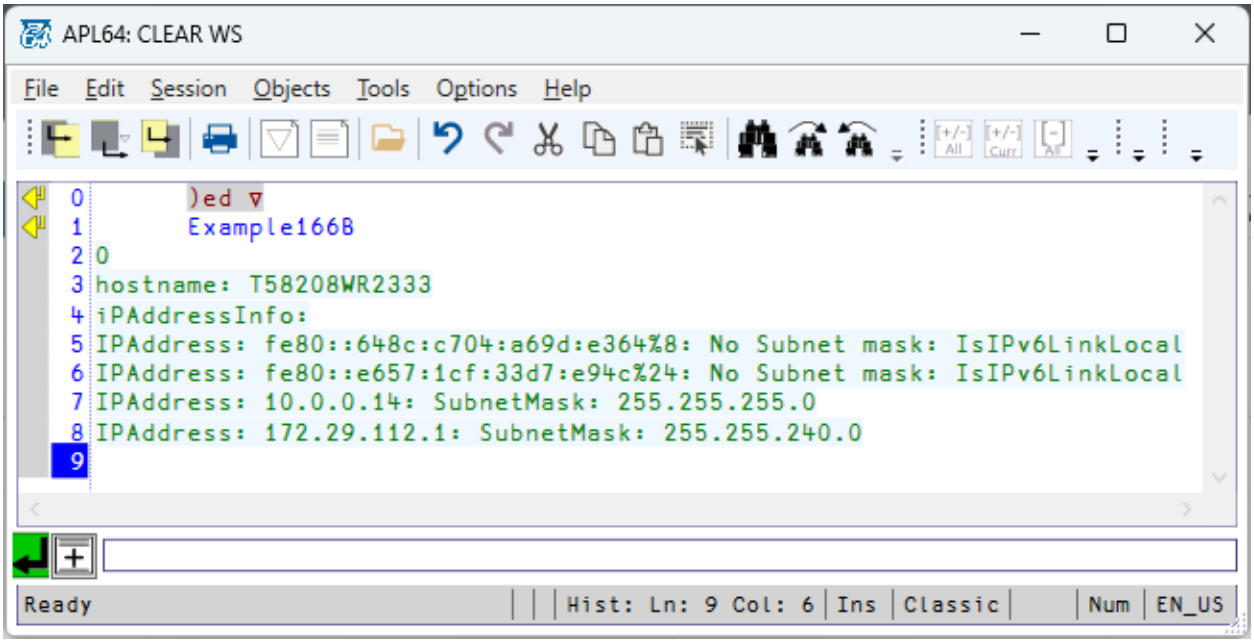
- hostName, a string variable
- ipEntry, an instance of the IPHostEntry type
- ipAddresses array of .Net IPAddress objects
- ipAddressInfo an array of strings analogous to an APL64 vector of text vectors
- GetSubnetMask3() method

```
Example166B;S
⍝ APLNext 20251020
S←««
using System;
using System.Text;
using System.Net;
```

```
using System.Net.NetworkInformation;
using System.Net.Sockets;
var hostName = Dns.GetHostName();
var ipEntry = Dns.GetHostEntry(hostName);
var ipAddresses = ipEntry.AddressList;
IPAddress GetSubnetMask3(IPAddress ipAddr)
{
 foreach (var ni in NetworkInterface.GetAllNetworkInterfaces())
  {
   foreach (var unicastIPAddressInformation in ni.GetIPProperties().UnicastAddresses)
   {
    if (unicastIPAddressInformation.Address.AddressFamily == AddressFamily.InterNetwork)
    {
     if (ipAddr.Equals(unicastIPAddressInformation.Address))
     {
      return unicastIPAddressInformation.IPv4Mask;
     }
    }
   }
  }
 throw new ArgumentException($"Cannot find subnetmask for IP address {ipAddr}");
}
string[] ipAddressInfo = new string[ipAddresses.Length];
for (var I = 0; I<ipAddresses.Length; I++)
 {
   var ipAddr = ipAddresses[I];
   var sb = new StringBuilder();
   sb.Append("IPAddress: " + ipAddr.ToString()+": ");
   if (ipAddr.IsIPv6LinkLocal)
     sb.Append("No Subnet mask: IsIPv6LinkLocal");
   else if (ipAddr.IsIPv6Multicast)
     sb.Append("No Subnet mask: IsIPv6Multicast");
   else if (ipAddr.IsIPv6SiteLocal)
     sb.Append("No Subnet mask: IsIPv6SiteLocal");
   else if (ipAddr.IsIPv6Teredo)
     sb.Append("No Subnet mask: IsIPv6Teredo");
   else
     sb.Append("SubnetMask: " + GetSubnetMask3(ipAddr).ToString());
   ipAddressInfo[I]= sb.ToString();
 }
 »»
□cself←'cse'□cse 'Init' 'System'
□cse 'Exec' S
'hostname: ',⍎□cse 'GetValue' 'hostName'
'iPAddressInfo:'
```

```
⊃>¨□cse 'GetValue' 'ipAddressInfo'
□cse 'Close'
```





Using the CSE with other .Net Languages

Example #167:

The CSE supports two main types of functionality:

- Execution (debugging, compiling and running) of text scripts that have been written using the syntax of the C# programming language. When the CSE 'Exec', 'ExecStmt' or 'ExecFile' method is used, the CSE:
    - Uses the .Net C# debugger and compiler to compile the script to the Common Intermediate Language (CIL) of .Net
    - Creates a memory-based .Net assembly which exists transiently while the instance of the CSE object exists
    - Runs any executable C# statements in the script on the Common Language Runtime (CLR) virtual machine of .Net

  For this functionality a C# language script (text file or APL variable) is created by the programmer. The script can be created directly in APL64 or using an ASCII- or Unicode-based text editor, such as Microsoft Notepad. Because the CSE will invoke the C# debugger and compiler on the programmer-provided script, Visual Studio is not required.

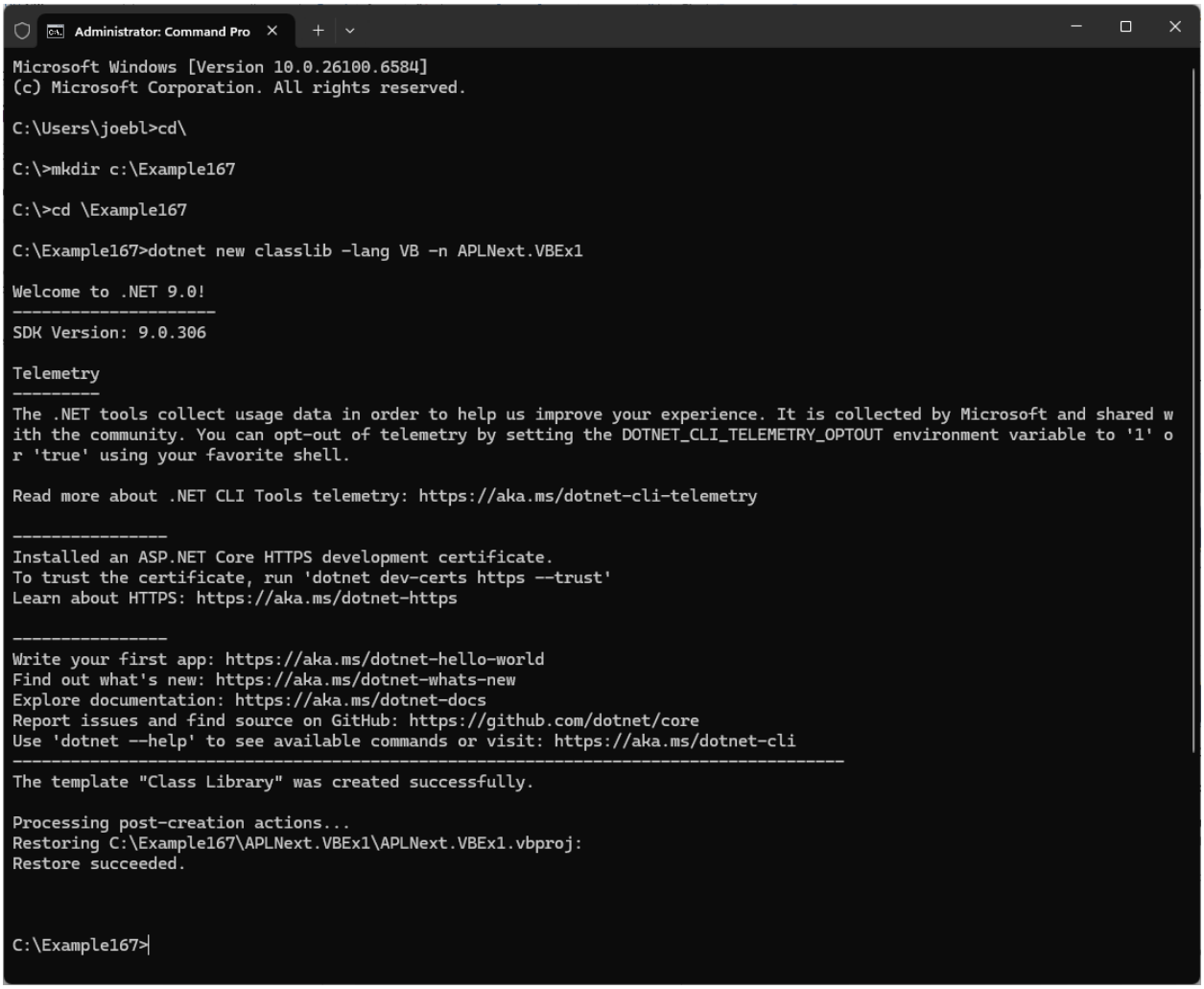
- Loading a previously-created .Net assembly which exists as physical .dll file using the CSE 'Init', 'LoadAssembly' or 'LoadAssemblyByName' methods. A .Net assembly defines .Net classes. Each .Net class includes the definition of its object model (methods, properties and events). Since all .Net programming languages emit CIL, a .Net assembly written in any .Net programming language, such as C#, VB.Net, F# or VisualAPL, can be used in a CSE script.

The functionality option requires a .Net assembly written in any .Net programming language and compiled to a physical .dll file generally using Visual Studio.

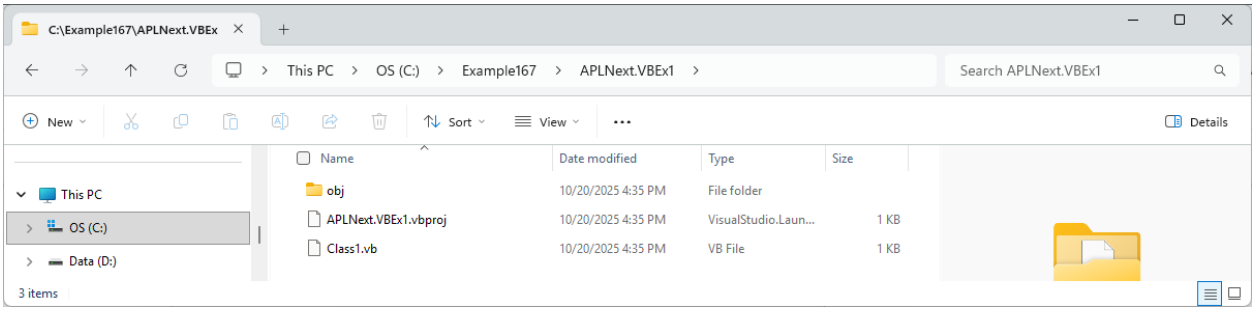
Example #167 will illustrate the second functionality using the '\CSE Code Samples\APLNext.VBEx1' Visual Studio solution which was written in VB.Net. This solution defines the 'Class1' .Net class with 'Input' and 'Output' properties and 'SqRt' function'.

The appropriate .Net SDK version for APL64 should be installed on the APL64 Developer version workstation. In this example the appropriate .Net SDK version is v9. Create the .Net library folders and files using the Windows Command Prompt:

```
cd\
mkdir c:\Example167
cd\ Example167
dotnet new classlib -lang VB -n APLNext.VBEx1
```

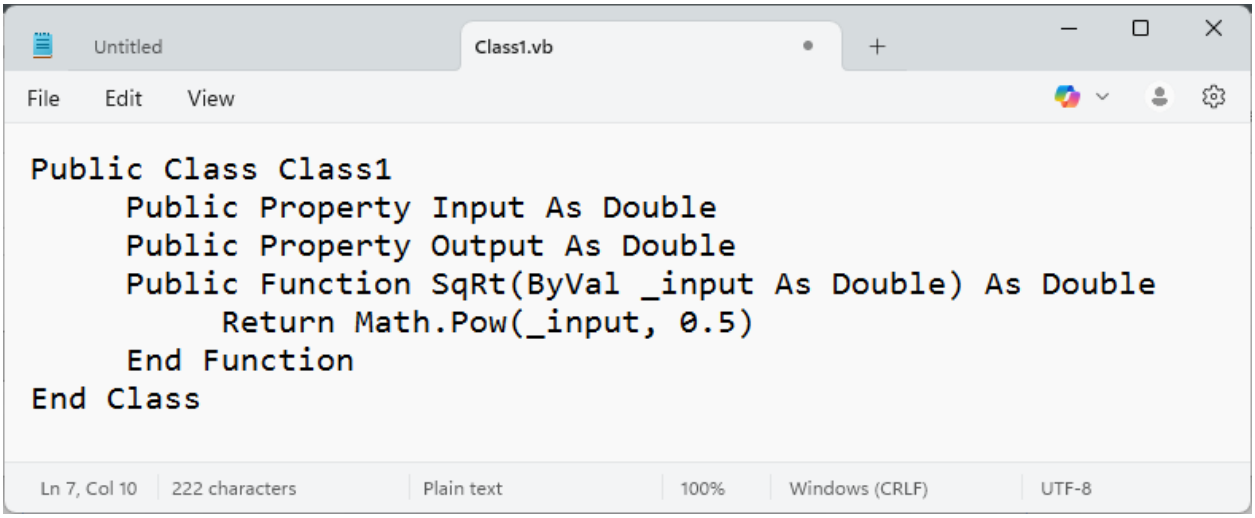


These commands create the c:\Example167\APLNext.VBEx1\ folder and, within that folder, created the VisualBasic project files:



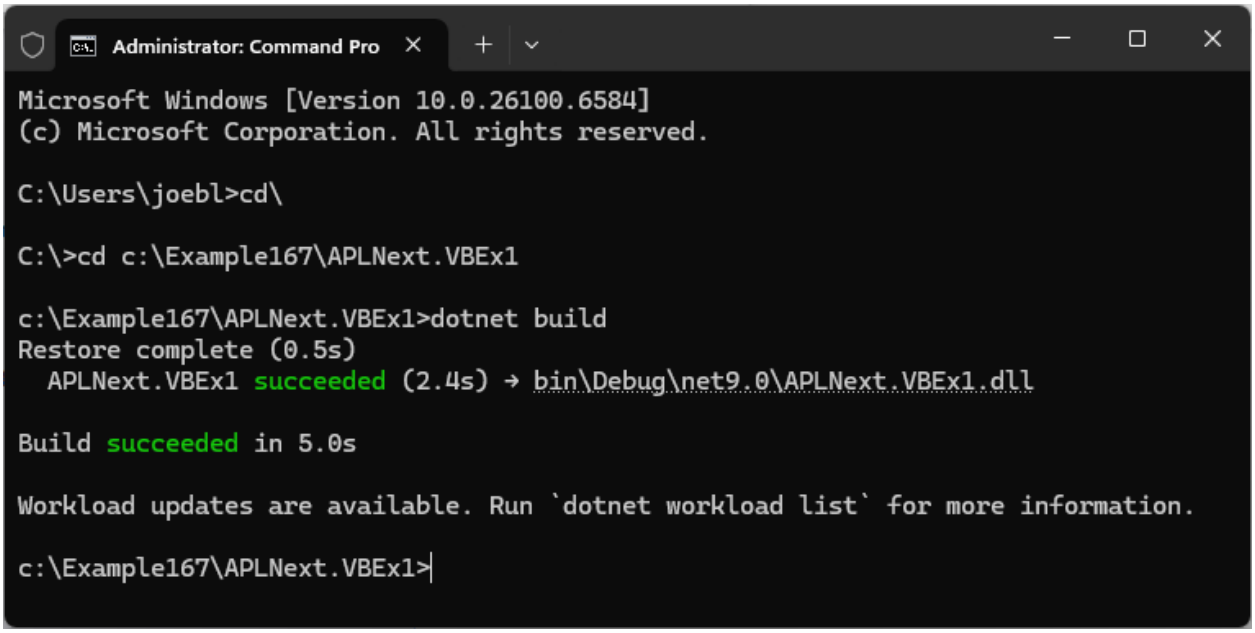
Using a text editor, in this case Windows Notepad, enter this source code into the Class1.vb file in the new folder and save the updated file.

```
Public Class Class1
    Public Property Input As Double
    Public Property Output As Double
    Public Function SqRt(ByVal _input As Double) As Double
        Return Math.Pow(_input, 0.5)
    End Function
End Class
```



Using the Windows Command Prompt, navigate to the c:\Example167\APLNext.VBEx1\ folder and build the library:

```
cd\
cd c:\Example167\APLNext.VBEx1
dotnet build
```



These commands create the APLNext.VBEx1.dll .Net assembly in the folder:
c:\Example167\APLNext.VBEx1 \bin\Debug\net9.0\

After compiling and saving the compiled code in Visual Studio, the 'APLNext.VBEx1.dll' .Net assembly exists as a file which can be used from APL64 using the CSE. This example illustrates:

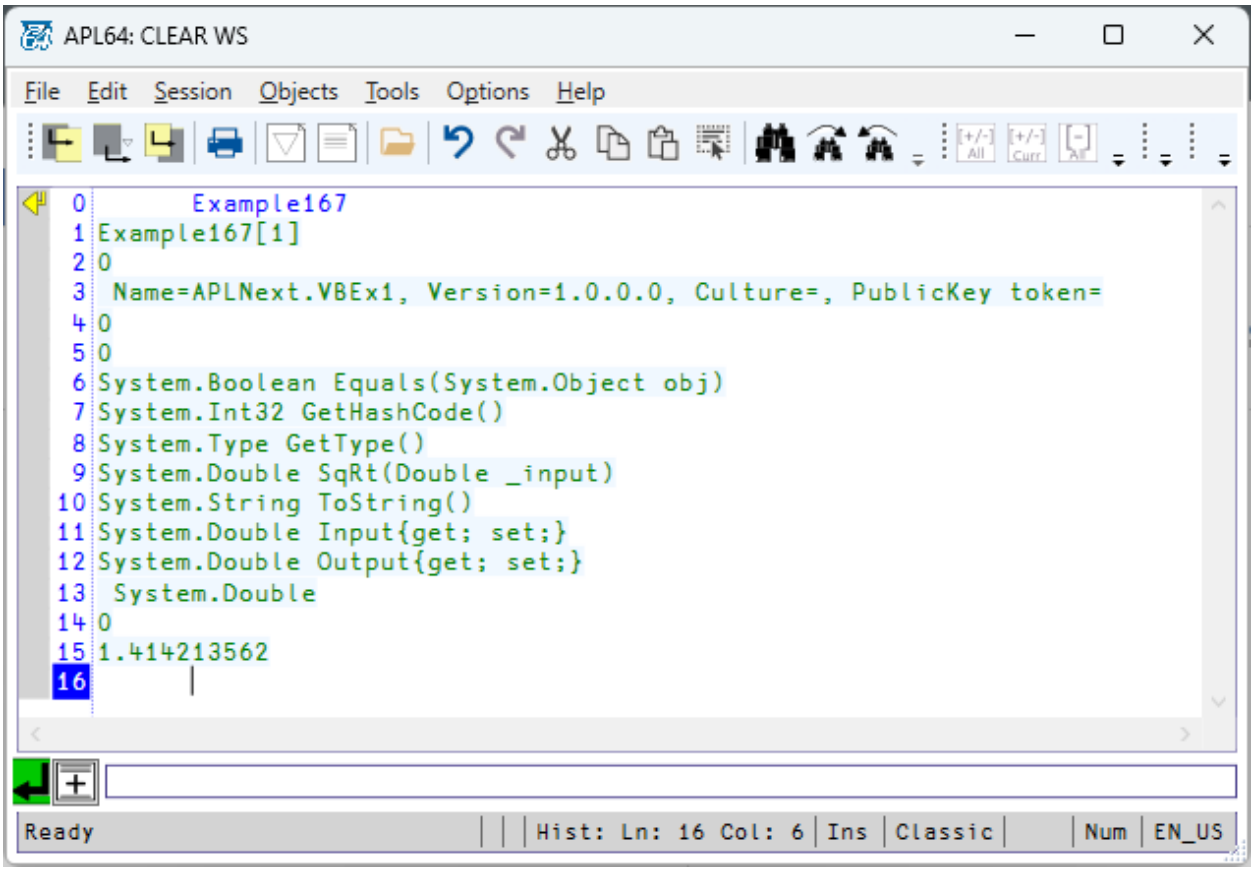
- Using the CSE 'LoadAssembly' method to create an instance of the 'APLNext.VBEx1.dll' .Net assembly in the CSE object.
- Using the CSE 'ExecStmt' to create an instance of the 'APLNext.VBEx1.Class1' .Net class
- Querying the methods and properties of the 'APLNext.VBEx1.Class1' using the CSE 'GetMethods' and 'GetProperties' methods
- Using the CSE 'SetValue', 'GetValue' and 'ExecStmt' methods to interact with that .Net class from APL64.

Create the Example167 function using this definition:

```
Example167;asmPath
 cself←'C' cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'
asmPath← 'c:\Example167\APLNext.VBEx1\bin\Debug\net9.0\APLNext.VBEx1.dll'
 cse 'LoadAssembly' asmPath
 cse 'ExecStmt' 'using APLNext.VBEx1;'
 cse 'ExecStmt' 'Class1 c1 = new Class1();'
⊃ cse 'GetMethods' 'c1'
⊃ cse 'GetProperties' 'c1'
 cse 'GetObjectType' 'c1.Input'
 cse 'SetValue' 'c1.Input' 2
```

```
]cse 'GetValue' 'c1.SqRt(c1.Input)'
]cse 'Close'
```

Run the Example167 function in APL64:



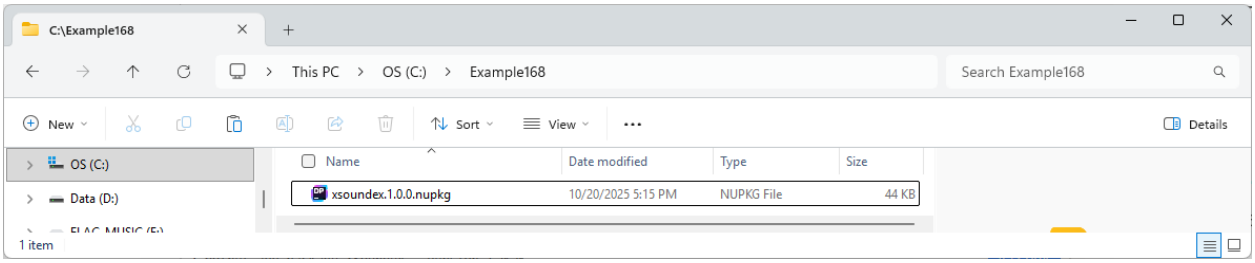
Soundex in .Net and APL64

Example #168:

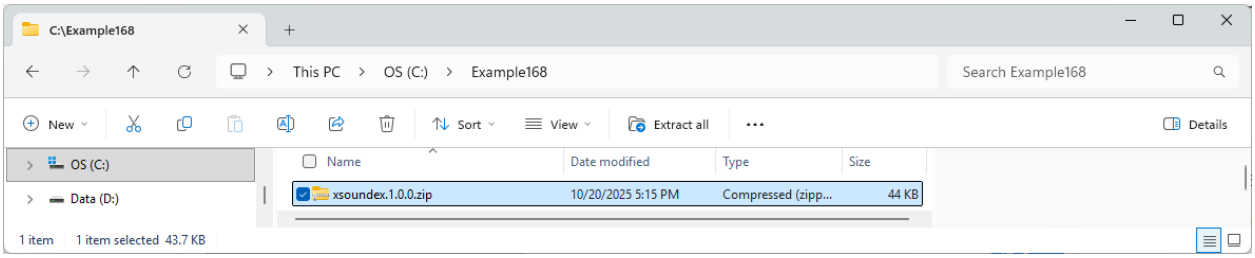
Soundex is an encoding, based on phonetics, converting a string representing a word into a string which may minimize minor spelling differences. Searching a large text or list for a specific word, ignoring minor spelling differences, is a typical use of Soundex encoding.

In Example168A the APL64 CSE uses an open source .Net implementation of Soundex.

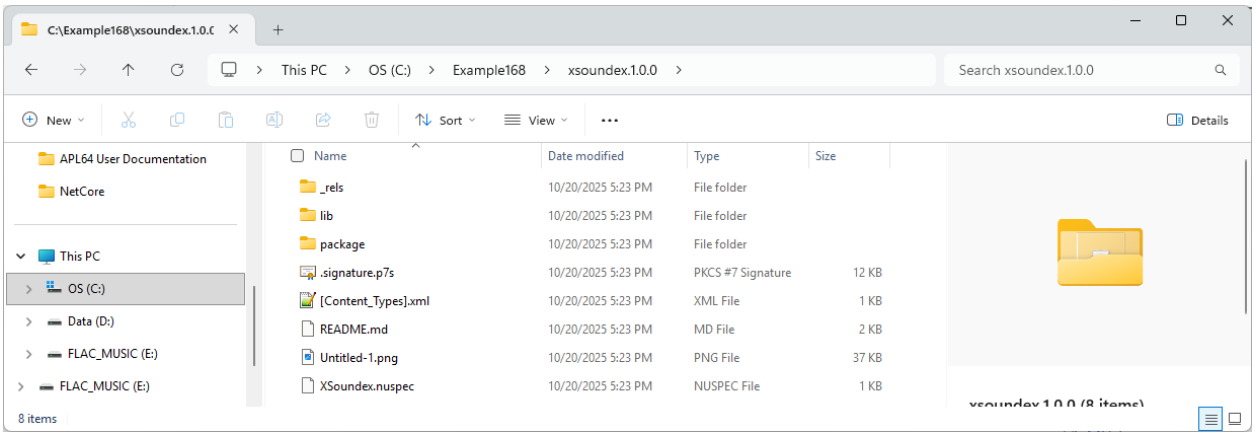
From the Nuget on-line repository (https://www.nuget.org/packages/XSoundex/1.0.0), download the XSoundex Nuget package to the c:\Example168 folder on the workstation:



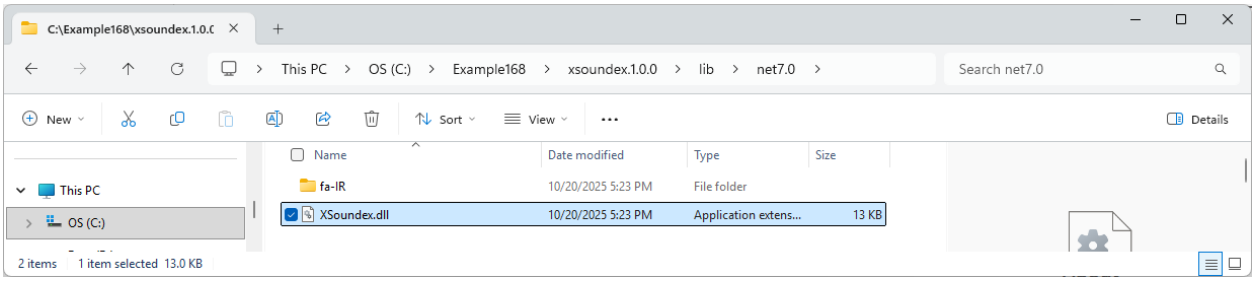
Rename the c:\Example168\ xsoundex.1.0.0.nupkg file to c:\Example167\ xsoundex.1.0.0.zip:



Unzip the c:\Example168\ xsoundex.1.0.0.zip file to the c:\Example167\xsoundex.1.0.0 folder:

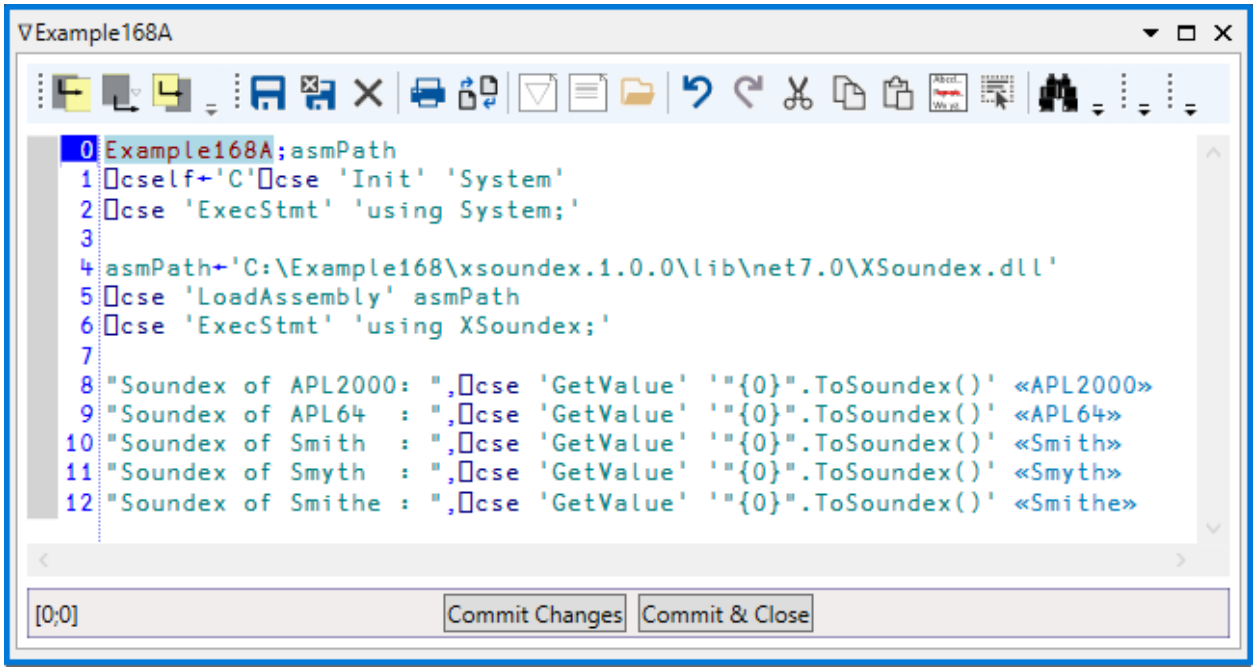


The XSoundex.dll .Net assembly is in the c:\Example168\xsoundex.1.0.0\lib\net7.0:

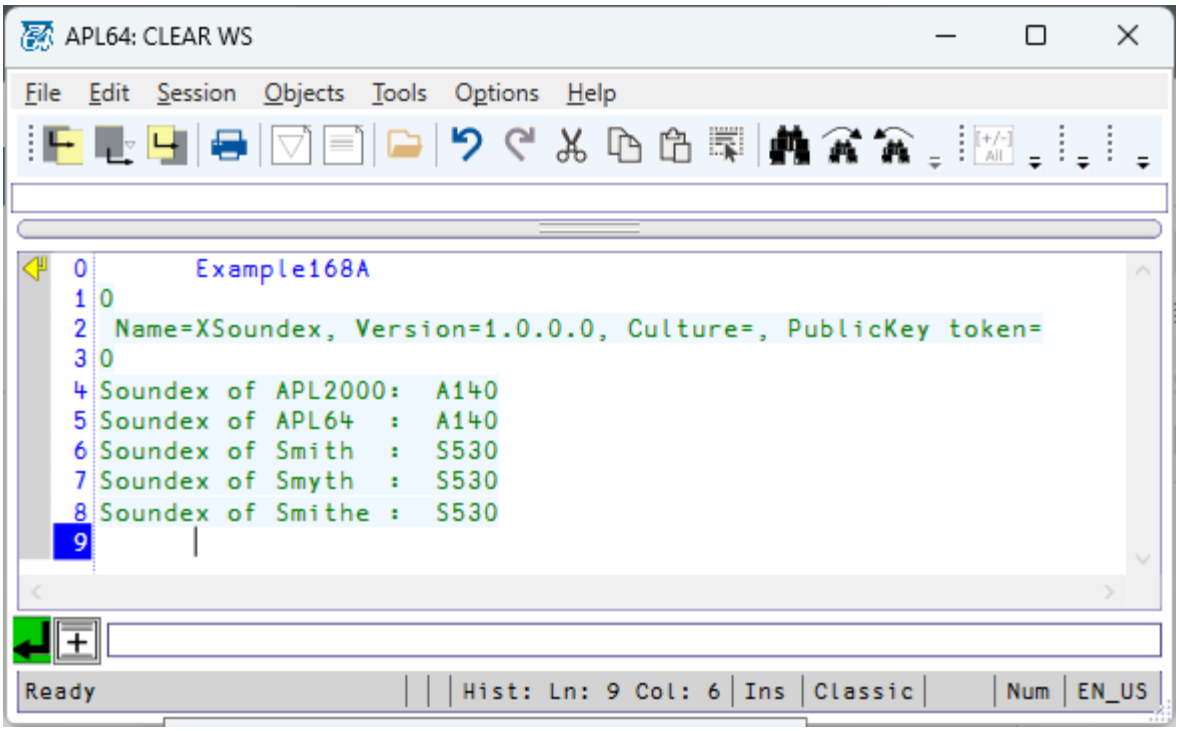


Example168A uses multiple executions of the String.ToSoundex() method to observe that three spellings of a name are phonetically very similar. Create the Example168A function using this definition:

```
Example168A;asmPath
 cself←'C' cse 'Init' 'System'
 cse 'ExecStmt' 'using System;'

asmPath←'C:\Example168\xsoundex.1.0.0\lib\net7.0\XSoundex.dll'
 cse 'LoadAssembly' asmPath
 cse 'ExecStmt' 'using XSoundex;'

"Soundex of APL2000: ", cse 'GetValue' '"{0}".ToSoundex()' «APL2000»
"Soundex of APL64  : ", cse 'GetValue' '"{0}".ToSoundex()' «APL64»
"Soundex of Smith  : ", cse 'GetValue' '"{0}".ToSoundex()' «Smith»
"Soundex of Smyth  : ", cse 'GetValue' '"{0}".ToSoundex()' «Smyth»
"Soundex of Smithe : ", cse 'GetValue' '"{0}".ToSoundex()' «Smithe»
```



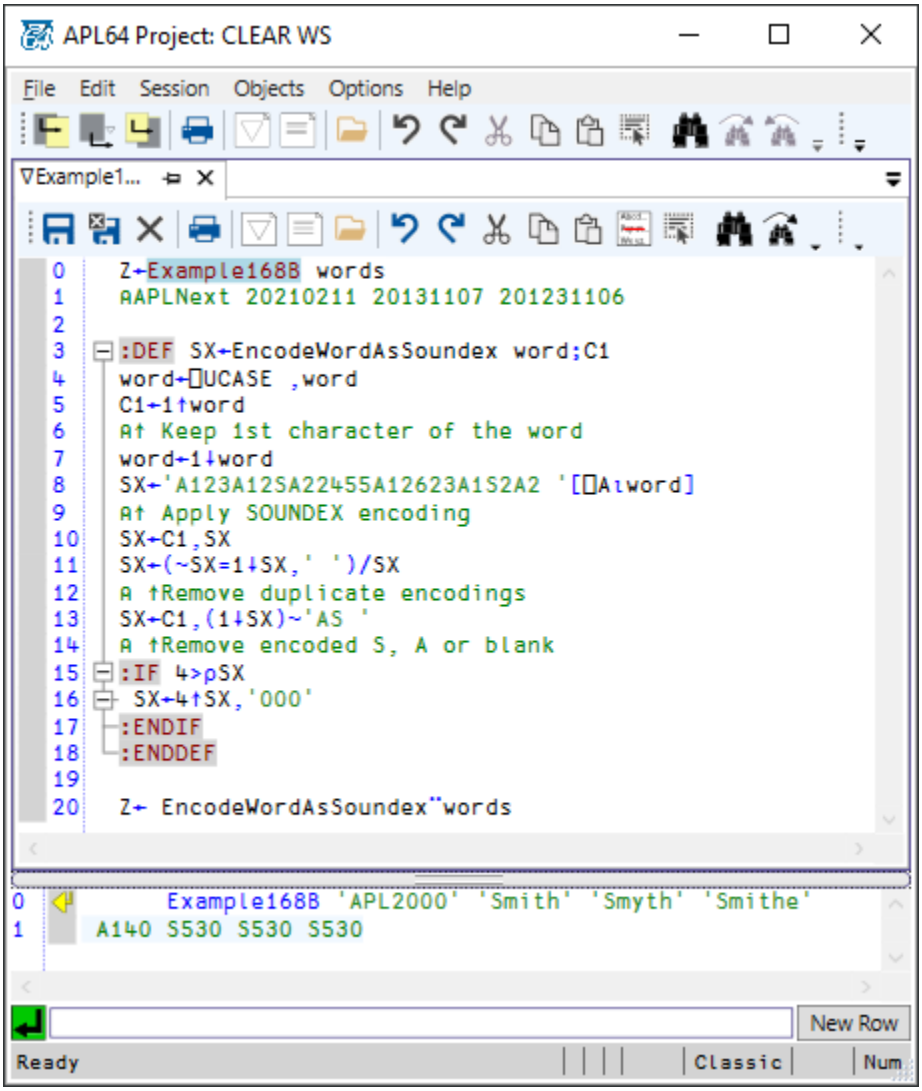
Run the Example168A function:



Example168B illustrates a direct implementation Soundex in APL64 without the .Net assembly or the CSE. This example also illustrates the use of an APL64 inner function, delineated by the :DEF/:ENDDEF control statements.

```
Z←Example168B words
⍝APLNext 20210211 20131107 201231106

:DEF SX←EncodeWordAsSoundex word;C1
word←⎕UCASE ,word
C1←1↑word
⍝↑ Keep 1st character of the word
word←1↓word
SX←'A123A12SA22455A12623A1S2A2 '[⎕Aɩword]
⍝↑ Apply SOUNDEX encoding
SX←C1,SX
SX←(~SX=1↓SX,' ')/SX
⍝ ↑Remove duplicate encodings
SX←C1,(1↓SX)~'AS '
⍝ ↑Remove encoded S, A or blank
:IF 4>⍴SX
 SX←4↑SX,'000'
:ENDIF
:ENDDEF
```

```
Z← EncodeWordAsSoundex¨words
```



Variable Precision Arithmetic Using Microsoft System.Numerics

Example #176

Overview

The mathematical operations performed by general purpose computers are based on fixed precision arithmetic, so that in some cases round-off error will prevent obtaining the desired results. Variable precision arithmetic libraries are available to support higher levels of precision to avoid this situation.

In .Net starting with version 4.5, Microsoft has added support for mathematical operations with variable precision in the System.Numerics namespace. Using the System.Numerics.BigInteger structure, mathematical operations can be performed on arbitrarily large numbers. The features of this namespace can be used via the CSE.

The value types available in APL64 include Boolean (1 bit per value), Integer (4 bytes per value), Double Precision Floating Point (8 bytes per value) and Character Vector (variable number of bytes). The precision (i.e. number of significant digits) afforded by each of the APL64 numeric value types is limited. Double precision floating point provides 16 digit precision.

Although character vector data may not initially be deemed suitable for storing numeric information, it can be used for that purpose because it supports a variable number of bytes per value.

While it is possible to develop an APL64-based tool set for arithmetic operations acting on numeric information stored in character vectors, such a library is already available without cost or license restrictions in the Microsoft .Net 4.0 System.Numerics.BigInteger namespace. This library may be efficiently employed using the APL64 □cse system function.

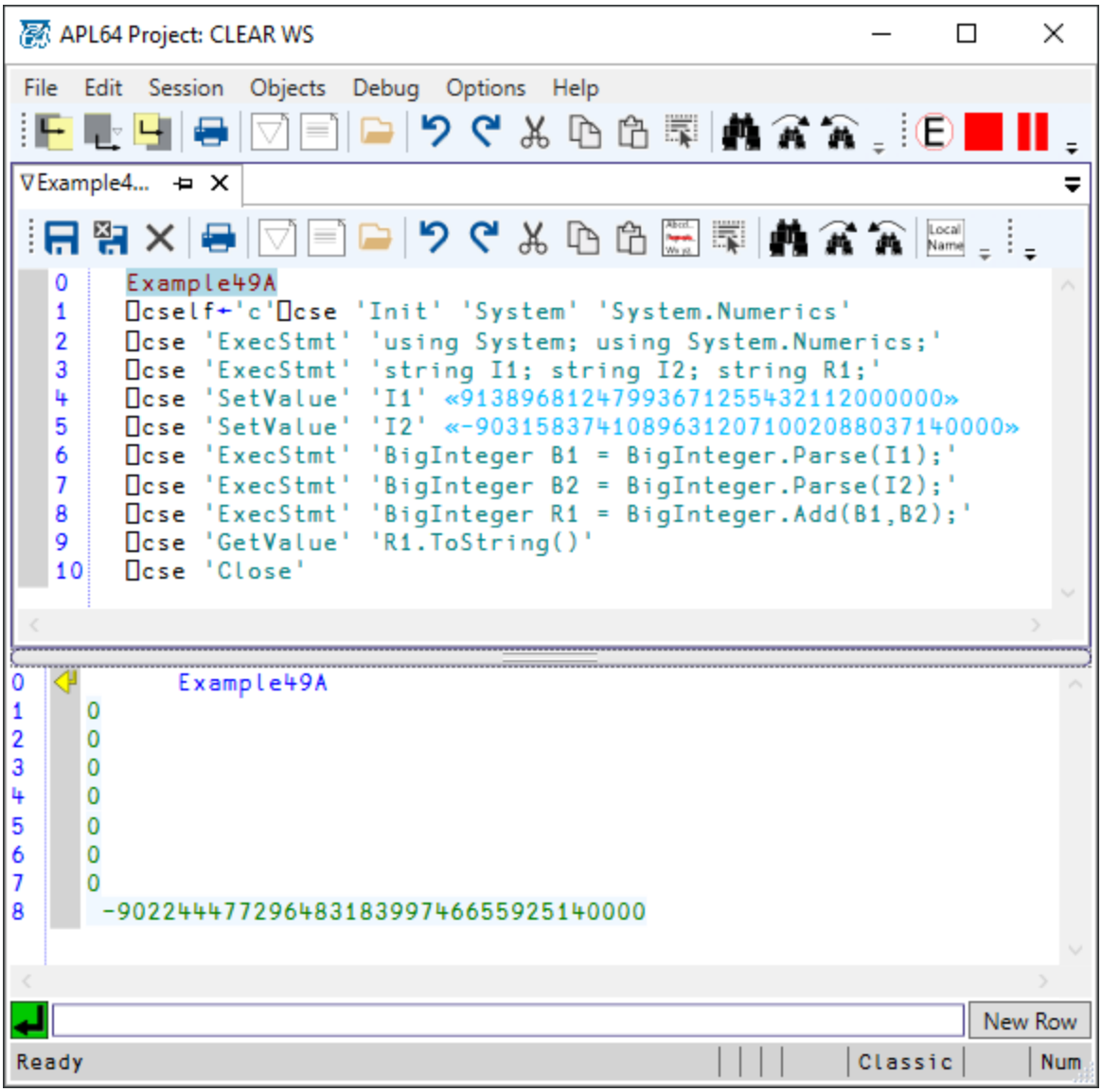
More System.Numerics Information

- Microsoft Developer Network
- Example using BigInteger for square root
- Example using BigInteger for image processing
- Examples using BigInteger for Fermat Primality Test, Jacobi Symbol Computation, Modal Exponentiation

Example49A

In this example the source values and the result value are impossible to represent precisely using APL64 numeric value types.

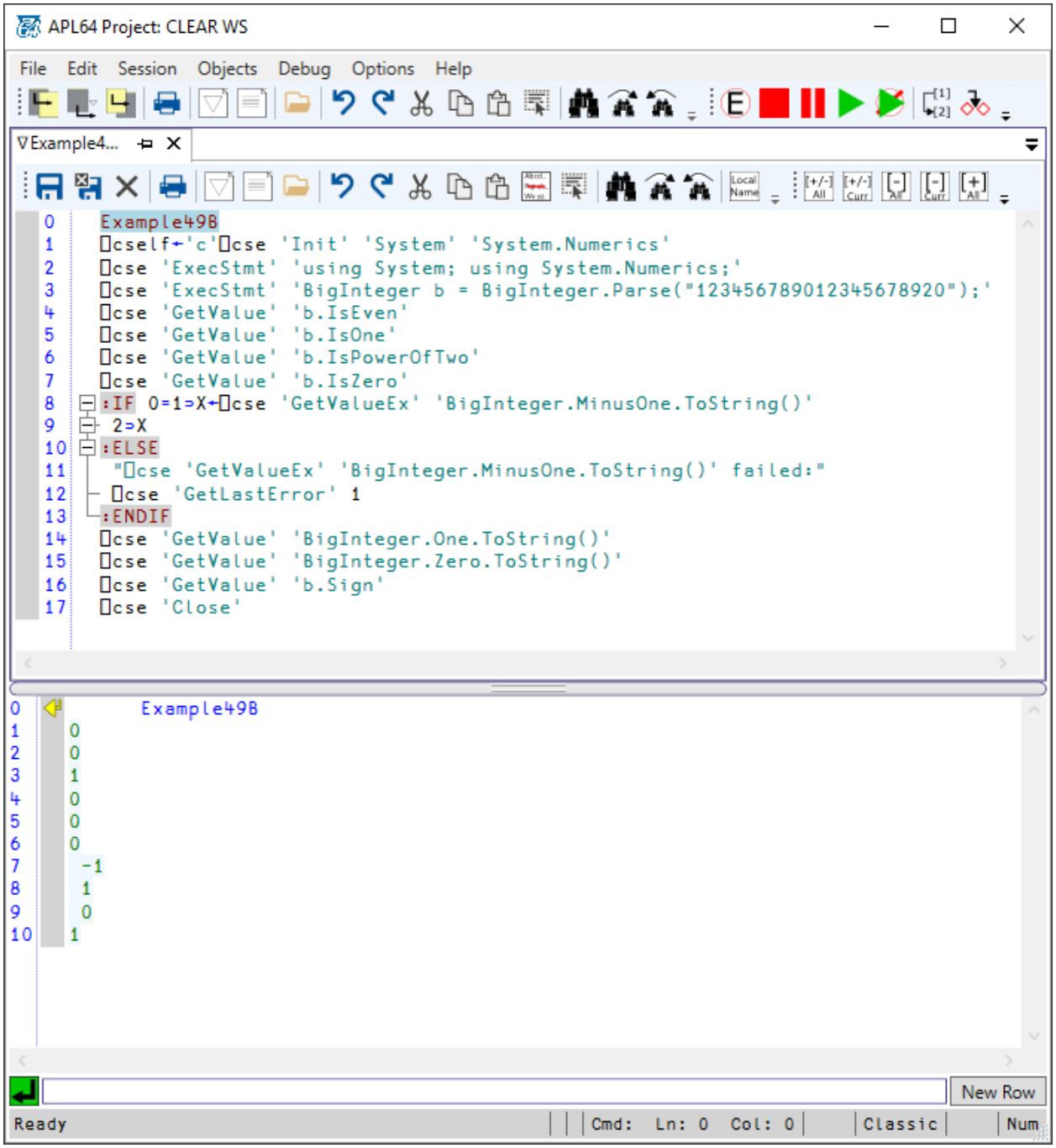
```
Example49A
□cself←'c'□cse 'Init' 'System' 'System.Numerics'
□cse 'ExecStmt' 'using System; using System.Numerics;'
□cse 'ExecStmt' 'string I1; string I2; string R1;'
□cse 'SetValue' 'I1' «91389681247993671255432112000000»
□cse 'SetValue' 'I2' «-90315837410896312071002088037140000»
□cse 'ExecStmt' 'BigInteger B1 = BigInteger.Parse(I1);'
□cse 'ExecStmt' 'BigInteger B2 = BigInteger.Parse(I2);'
□cse 'ExecStmt' 'BigInteger R1 = BigInteger.Add(B1,B2);'
□cse 'GetValue' 'R1.ToString()'
□cse 'Close'
```



Many Useful Methods and Operators are supported

```
Example49B
⎕cself←'c'⎕cse 'Init' 'System' 'System.Numerics'
⎕cse 'ExecStmt' 'using System; using System.Numerics;'
⎕cse 'ExecStmt' 'BigInteger b = BigInteger.Parse("123456789012345678920");'
⎕cse 'GetValue' 'b.IsEven'
⎕cse 'GetValue' 'b.IsOne'
⎕cse 'GetValue' 'b.IsPowerOfTwo'
⎕cse 'GetValue' 'b.IsZero'
:IF 0=1⊃X←⎕cse 'GetValueEx' 'BigInteger.MinusOne.ToString()'
 2⊃X
:ELSE
 "⎕cse 'GetValueEx' 'BigInteger.MinusOne.ToString()' failed:"
```

```
 ⎕cse 'GetLastError' 1
:ENDIF
⎕cse 'GetValue' 'BigInteger.One.ToString()'
⎕cse 'GetValue' 'BigInteger.Zero.ToString()'
⎕cse 'GetValue' 'b.Sign'
⎕cse 'Close'
```

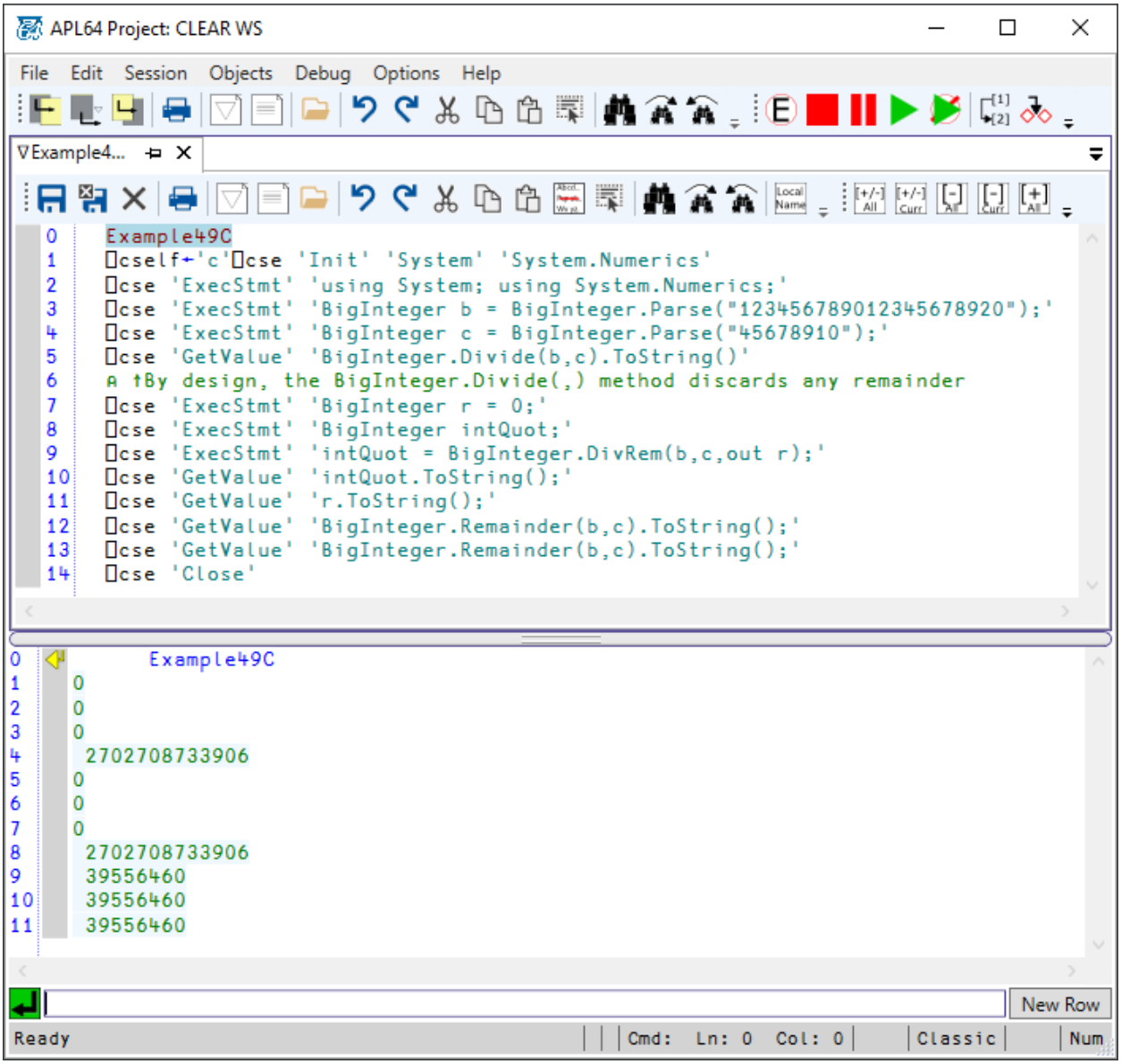


Integer.Divide, BigInteger.Remainder and BigInteger.DivRem methods

The BigInteger.Divide(,) method returns the integral portion of the division. The BigInteger.Remainder(,) method returns the integral remainder of the division. The BigInteger.DivRem(,,out) method provides both the integral quotient and the integral remainder of the divide operation.

Example #49C

```
BigIntegerDivide
Example49C
□cself←'c'□cse 'Init' 'System' 'System.Numerics'
□cse 'ExecStmt' 'using System; using System.Numerics;'
□cse 'ExecStmt' 'BigInteger b = BigInteger.Parse("123456789012345678920");'
□cse 'ExecStmt' 'BigInteger c = BigInteger.Parse("45678910");'
□cse 'GetValue' 'BigInteger.Divide(b,c).ToString()'
⍝ ↑By design, the BigInteger.Divide(,) method discards any remainder
□cse 'ExecStmt' 'BigInteger r = 0;'
□cse 'ExecStmt' 'BigInteger intQuot;'
□cse 'ExecStmt' 'intQuot = BigInteger.DivRem(b,c,out r);'
□cse 'GetValue' 'intQuot.ToString();'
□cse 'GetValue' 'r.ToString();'
□cse 'GetValue' 'BigInteger.Remainder(b,c).ToString();'
□cse 'GetValue' 'BigInteger.Remainder(b,c).ToString();'
□cse 'Close'
```



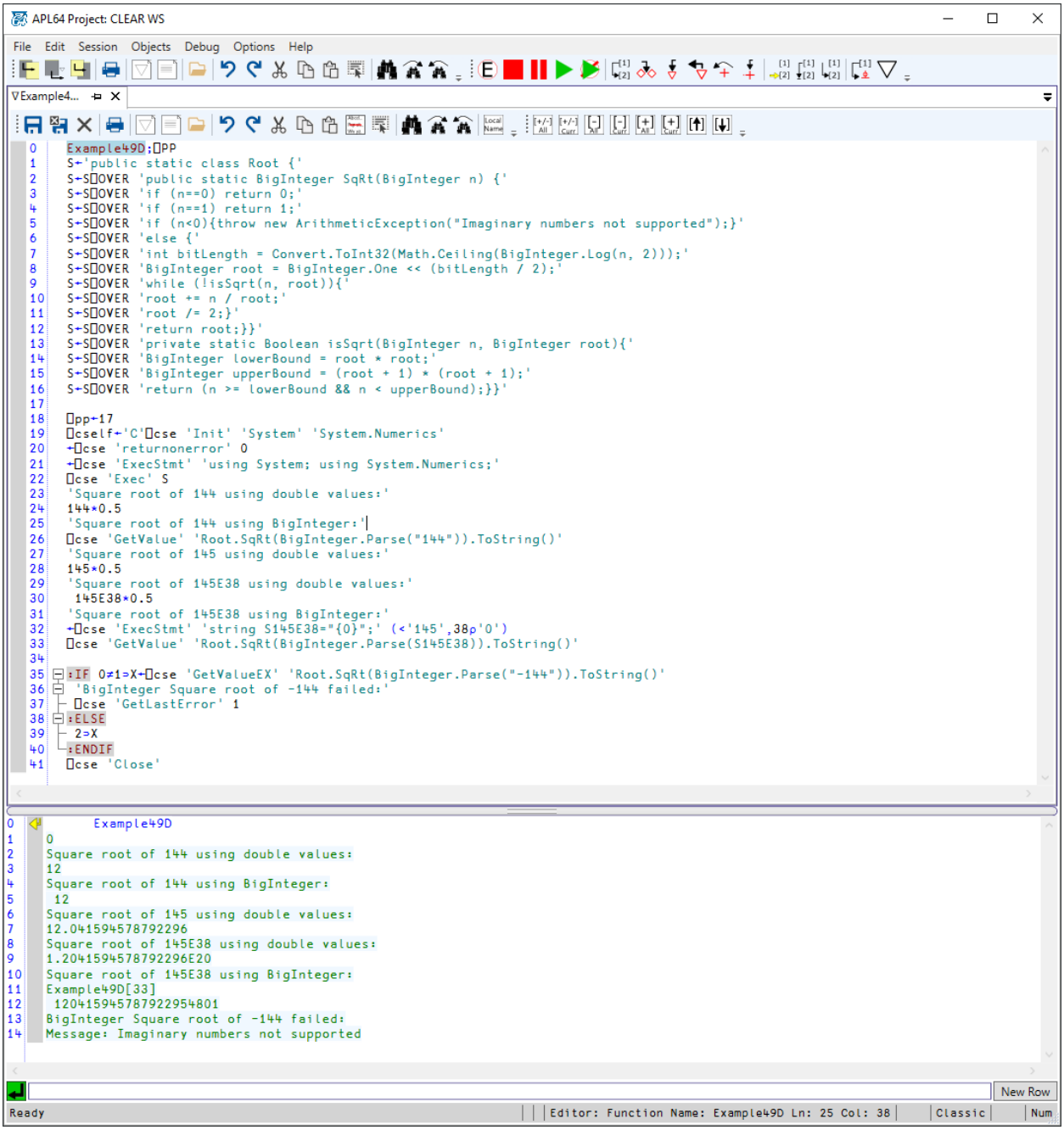
BigInteger.Pow(,) method supports integer exponentiation

The BigInteger.Pow(,) method supports only integer exponentiation, so custom algorithms are necessary for non-integral exponents. One possible square root (exponent 0.5) algorithm is illustrated here with scaling used to increase the precision as desired.

Example #49D

```
Example49D;⎕PP
S←'public static class Root {'
S←S⎕OVER 'public static BigInteger SqRt(BigInteger n) {'
S←S⎕OVER 'if (n==0) return 0;'
S←S⎕OVER 'if (n==1) return 1;'
S←S⎕OVER 'if (n<0){throw new ArithmeticException("Imaginary numbers not supported");}'
S←S⎕OVER 'else {'
```

```
S←S⎕OVER 'int bitLength = Convert.ToInt32(Math.Ceiling(BigInteger.Log(n, 2)));'
S←S⎕OVER 'BigInteger root = BigInteger.One << (bitLength / 2);'
S←S⎕OVER 'while (!isSqrt(n, root)){'
S←S⎕OVER 'root += n / root;'
S←S⎕OVER 'root /= 2;}'
S←S⎕OVER 'return root;}}'
S←S⎕OVER 'private static Boolean isSqrt(BigInteger n, BigInteger root){'
S←S⎕OVER 'BigInteger lowerBound = root * root;'
S←S⎕OVER 'BigInteger upperBound = (root + 1) * (root + 1);'
S←S⎕OVER 'return (n >= lowerBound && n < upperBound);}}'

⎕pp←17
⎕cself←'C'⎕cse 'Init' 'System' 'System.Numerics'
←⎕cse 'returnonerror' 0
←⎕cse 'ExecStmt' 'using System; using System.Numerics;'
⎕cse 'Exec' S
'Square root of 144 using double values:'
144*0.5
'Square root of 144 using BigInteger:'
⎕cse 'GetValue' 'Root.SqRt(BigInteger.Parse("144")).ToString()'
'Square root of 145 using double values:'
145*0.5
'Square root of 145E38 using double values:'
 145E38*0.5
'Square root of 145E38 using BigInteger:'
←⎕cse 'ExecStmt' 'string S145E38="{0}";' (<'145',38⍴'0'S
⎕cse 'GetValue' 'Root.SqRt(BigInteger.Parse(s145E38)).ToString()'

:IF 0≠1⊃X←⎕cse 'GetValueEX' 'Root.SqRt(BigInteger.Parse("-144")).ToString()'
 'BigInteger Square root of -144 failed:'
 ⎕cse 'GetLastError' 1
:ELSE
 2⊃X
:ENDIF
⎕cse 'Close'
```



Using System.Text.RegularExpressions with the CSE

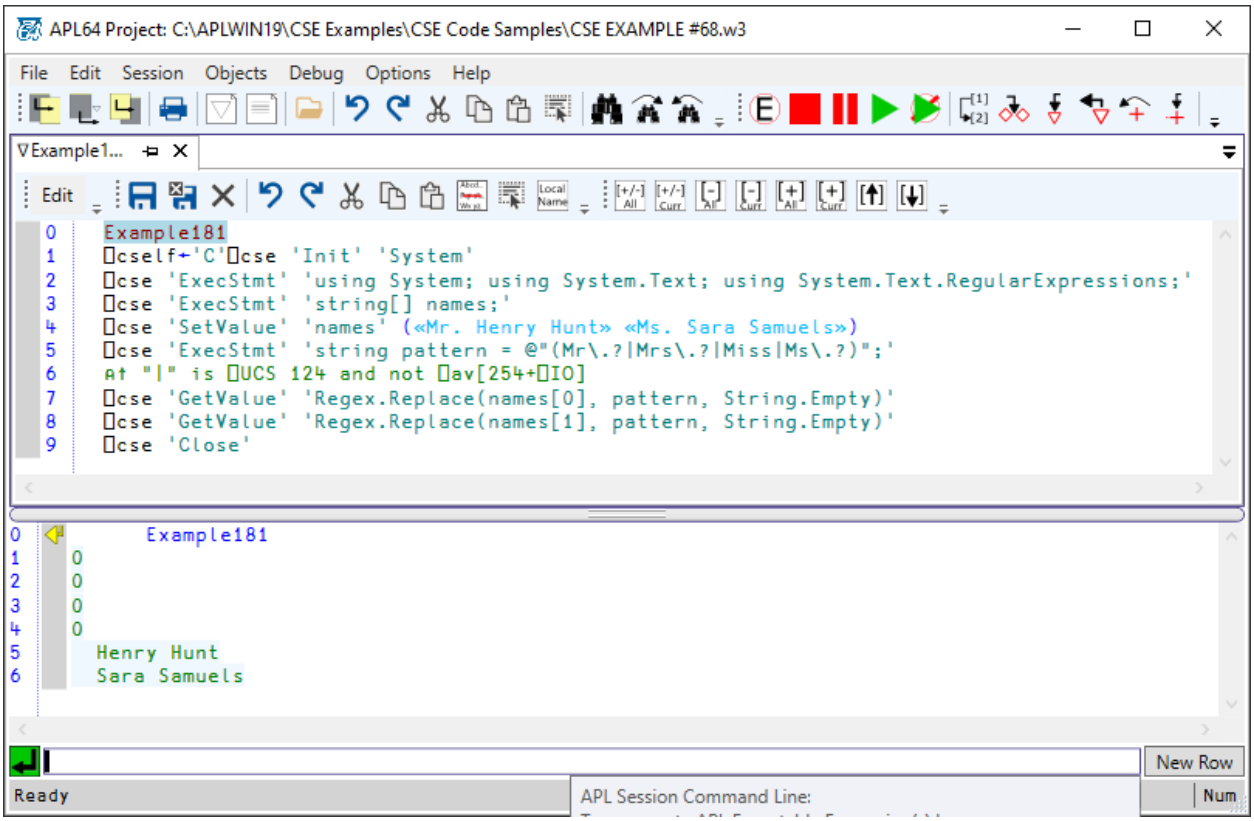
The .Net regular expressions engine supports the processing of text via pattern matching. This engine can identify text matches, edit, replace or delete text substrings. This engine uses the regular expression language to define the pattern to match. The pattern can contain text literals, operators and other constructs. For more information refer to:

http://msdn.microsoft.com/en-us/library/hs600312(v=vs.110).aspx http://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx http://msdn.microsoft.com/en-us/library/01escwtf(v=vs.110).aspx

Example #181: Delete selected text in an array of APL64 text vectors

In this example the .Net 'String.Empty' type replaces selected text in an array of APL64 text vectors.

```
Example181
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System; using System.Text; using System.Text.RegularExpressions;'
⎕cse 'ExecStmt' 'string[] names;'
⎕cse 'SetValue' 'names' («Mr. Henry Hunt» «Ms. Sara Samuels»)
⎕cse 'ExecStmt' 'string pattern = @"(Mr\.?|Mrs\.?|Miss|Ms\.?)";'
⍝↑ "|" is ⎕UCS 124 and not ⎕av[254+⎕IO]
⎕cse 'GetValue' 'Regex.Replace(names[0], pattern, String.Empty)'
⎕cse 'GetValue' 'Regex.Replace(names[1], pattern, String.Empty)'
⎕cse 'Close'
```



Example 183: Find duplicated text in a source text

In this example the .Net Regex.Matches() method is used to identify the value and index of repeated text in a source text vector. For further information study the properties of the .Net MatchCollection object.

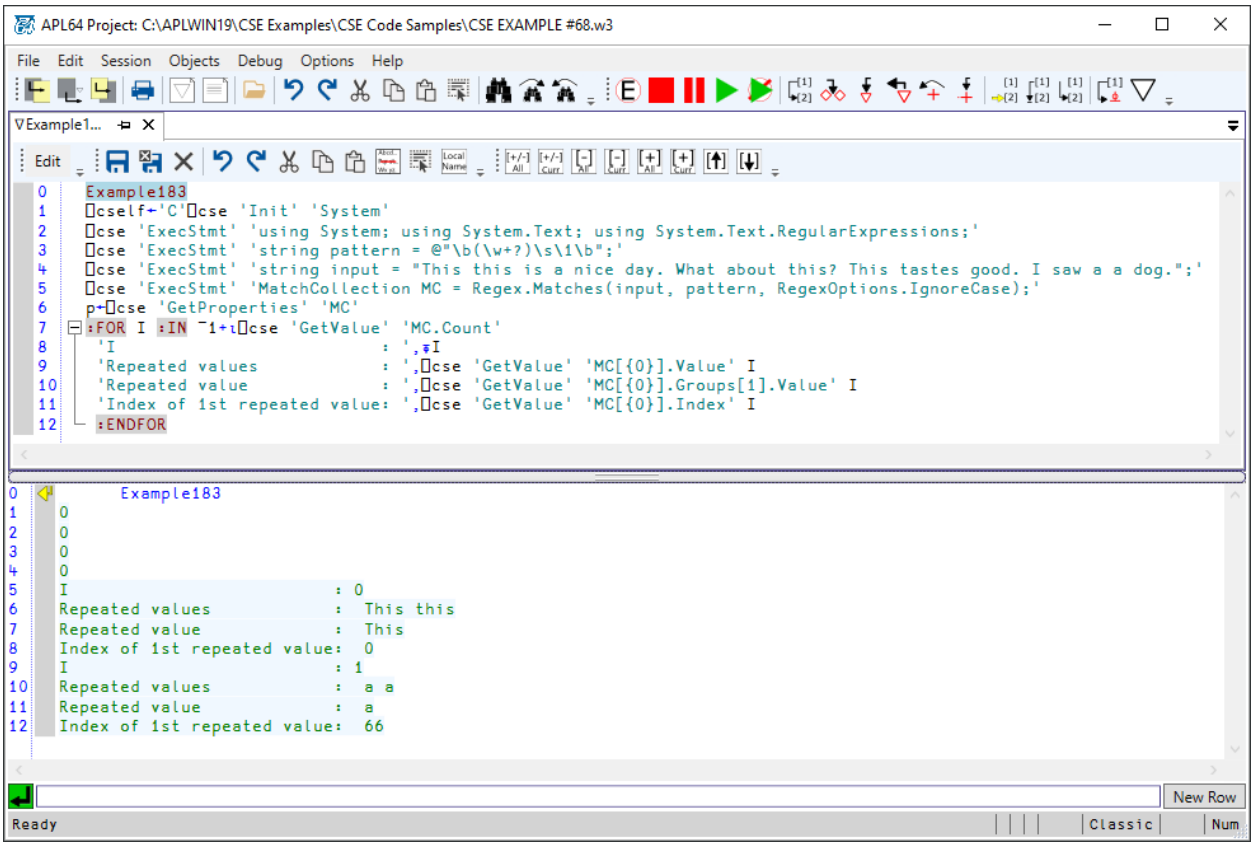
```
Example183
⎕cself←'C'⎕cse 'Init' 'System'
⎕cse 'ExecStmt' 'using System; using System.Text; using System.Text.RegularExpressions;'
⎕cse 'ExecStmt' 'string pattern=@"\b(\w+?)\s\1\b";'
⎕cse 'ExecStmt' 'string input="This this is a nice day. What about this? This tastes good. I saw a a dog.";'
⎕cse 'ExecStmt' 'MatchCollection MC = Regex.Matches(input, pattern, RegexOptions.IgnoreCase);'
```

```
p←□cse 'GetProperties' 'MC'
:FOR I :IN ¯1+⍳□cse 'GetValue' 'MC.Count'
 'I                        : ', ⍕I
 'Repeated values          : ',□cse 'GetValue' 'MC[{0}].Value' I
 'Repeated value           : ',□cse 'GetValue' 'MC[{0}].Groups[1].Value' I
 'Index of 1st repeated value: ',□cse 'GetValue' 'MC[{0}].Index' I
 :ENDFOR
```

Example183

```
□pw←150
□cself←'C'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System; using System.Text; using System.Text.RegularExpressions;'
□cse 'ExecStmt' 'string pattern = @"\b(\w+?)\s\1\b";'
□cse 'ExecStmt' 'string input = "This this is a nice day. What about this? This tastes good. I saw a a dog.";'
□cse 'ExecStmt' 'MatchCollection MC = Regex.Matches(input, pattern, RegexOptions.IgnoreCase);'
'Repeated values       : ',□cse 'GetValue' 'MC[1].Value'
'Repeated value        : ',□cse 'GetValue' 'MC[1].Groups[1].Value' 'Index of repeated value: ',□cse 'GetValue' 'MC[1].Index'
```

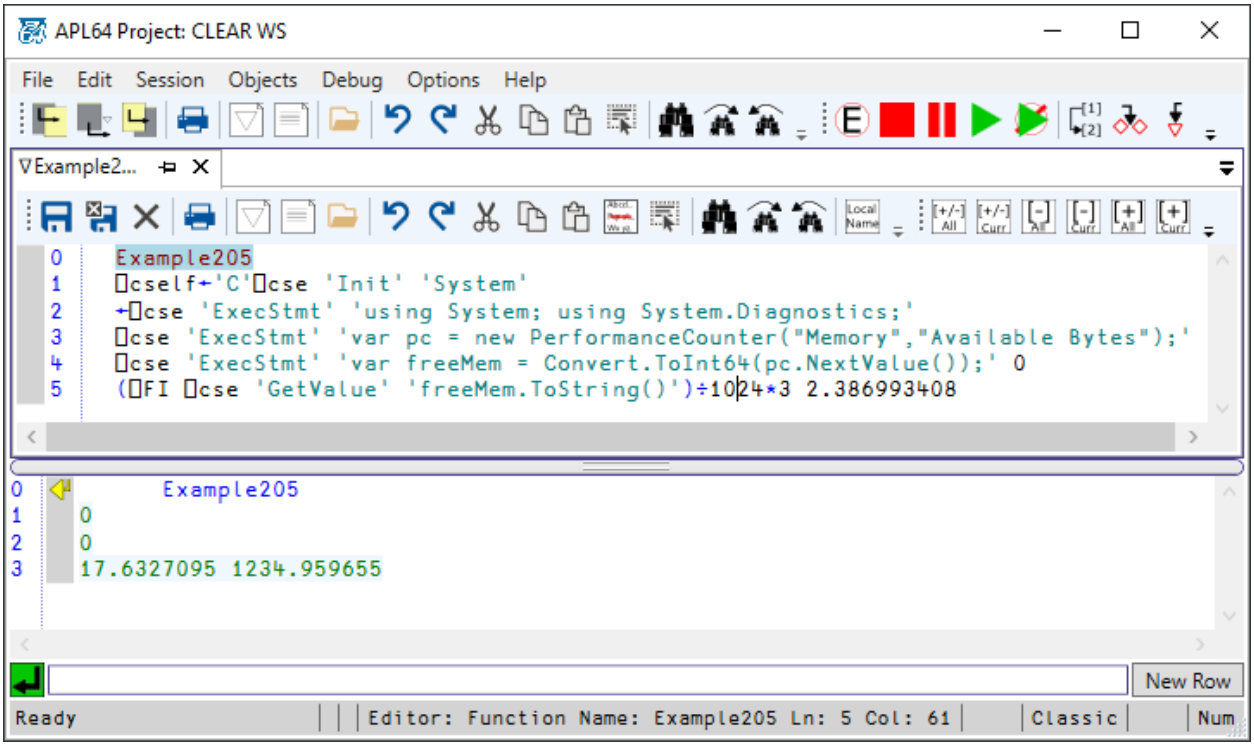
APL64 session when the 'Example183' function is executed:



Obtain the Free Memory Available on the Workstation

Example #205:

```
Example205
⍞←'C'⍞cse 'Init' 'System'
←⍞cse 'ExecStmt' 'using System; using System.Diagnostics;'
⍞cse 'ExecStmt' 'var pc = new PerformanceCounter("Memory","Available Bytes");'
⍞cse 'ExecStmt' 'var freeMem = Convert.ToInt64(pc.NextValue());' 0
(⍞FI ⍞cse 'GetValue' 'freeMem.ToString()')÷1024*3 2.386993408
```



For more information in .Net 'performance counters' go here:

- https://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter(v=vs.110).aspx
- https://msdn.microsoft.com/en-us/library/w8f5kw2e(v=vs.110).aspx

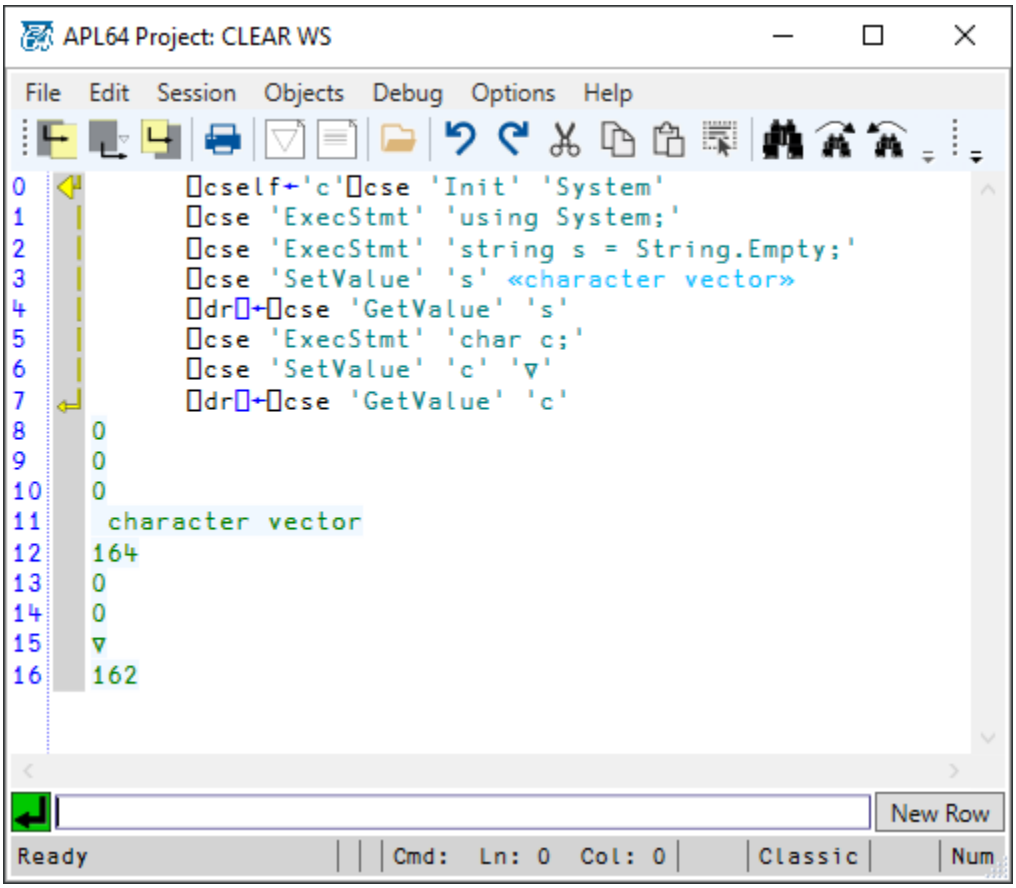
SetValue/GetValue: Text scalar

Example #103: SetValue/GetValue: Text scalar

This example illustrates that the .Net data type of the programmer-selected target container for scalar character data provided by APL64 affects the receipt on the CSE side and return of the data to the APL side.

If the .Net target data type is String, then the APL64 programmer should provide an APL64 string value when using SetValue. If the .Net target data type is Char, then the APL64 programmer should provide a character data type when using SetValue.

```
Example103
□cself←'c'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'string s = String.Empty;'
□cse 'SetValue' 's' «character vector»
□dr□←□cse 'GetValue' 's'
□cse 'ExecStmt' 'char c;'
□cse 'SetValue' 'c' '∇'
□dr□←□cse 'GetValue' 'c'
□cse 'Close'
```



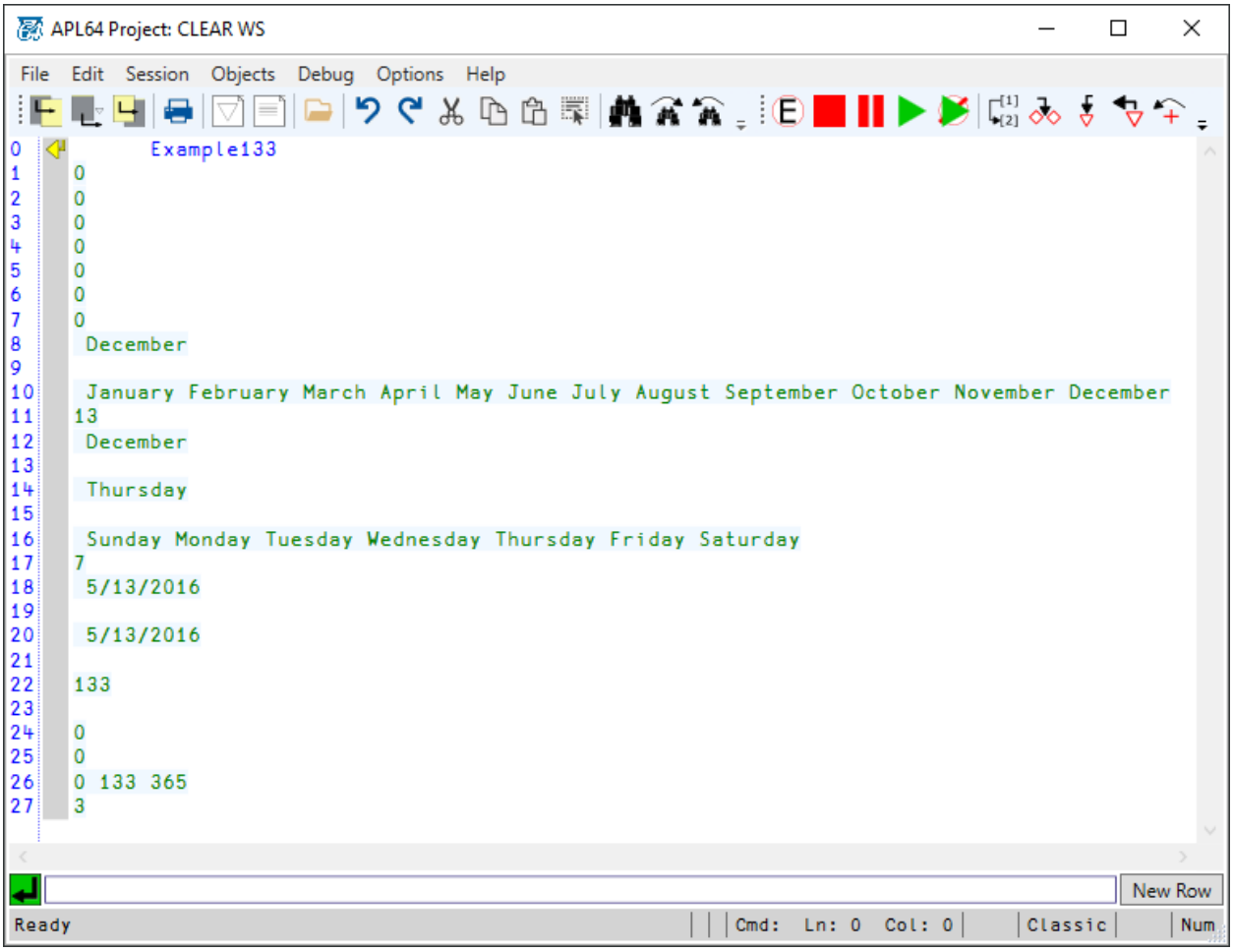
Example #133: Define C# Methods

This example illustrates the definition of C# methods without a class container.

```
Example133
□cself←'c'□cse 'Init' 'System'
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'using System.Globalization;'

S1←'DateTime DateStringToDateTime(string dateString){'
```

```
S1←S1,'return DateTime.ParseExact(dateString,"yyyyMMdd",CultureInfo.InvariantCulture);}'
⎕cse 'ExecStmt' S1
⍝ ↑Define the DateStringToDateTime() method using script S1

S2←'DateTime DateIntToDateTime(Int32 dateInt){'
S2←S2,'return DateStringToDateTime(dateInt.ToString());}'
⎕cse 'ExecStmt' S2
⍝ ↑Define the DateIntToDateTime() method using script S2

S3←'DateTime[] DateIntsToDateTimes(Int32[] dateInts){'
S3←S3,'DateTime[] dateTimes = new DateTime[dateInts.Length];'
S3←S3,'for(Int32 I = 0;I<dateInts.Length;I++) {dateTimes[I]=DateIntToDateTime(dateInts[I]);}'
S3←S3,'return dateTimes;}'
⎕cse 'ExecStmt' S3
⍝ ↑Define the DateIntsToDateTimes() method using script S3

S4←'Int32 DayDiff(Int32 valDate, Int32 dateInt){'
S4←S4,'return(Int32)DateIntToDateTime(dateInt).Subtract(DateIntToDateTime(valDate)).TotalDays;}'
⎕cse 'ExecStmt' S4
⍝ ↑Define the DayDiff() method using script S4

S5←'Int32[] DayDiffs(Int32 valDate, Int32[] dateInts){'
S5←S5,'Int32[] dayDiffs = new Int32[dateInts.Length];'
S5←S5,'for(Int32 I=0;I<dateInts.Length;I++) {dayDiffs[I]=DayDiff(valDate, dateInts[I]);}'
S5←S5,'return dayDiffs;}'
⎕cse 'ExecStmt' S5
⍝ ↑Define the DayDiffs() method using script S5

ρ⎕←⎕cse 'GetValue' 'CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName({0})' 12
ρ⎕←⊃⎕cse 'GetValue' 'DateTimeFormatInfo.CurrentInfo.MonthNames'
ρ⎕←⎕cse'GetValue'
'CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(DateTime.Parse("{0}").Month)'
'12/31/2015'
ρ⎕←⎕cse'GetValue'
'CultureInfo.CurrentCulture.DateTimeFormat.GetDayName(DateTime.Parse("{0}").DayOfWeek)'
'12/31/2015'
ρ⎕←⊃⎕cse 'GetValue' 'DateTimeFormatInfo.CurrentInfo.DayNames'
ρ⎕←⎕cse 'GetValue' 'DateStringToDateTime("{0}").ToShortDateString()' '20160513'
ρ⎕←⎕cse 'GetValue' 'DateIntToDateTime({0}).ToShortDateString()' 20160513
ρ⎕←⎕cse 'GetValue' 'DayDiff({0},{1})' 20160101 20160513

⎕cse 'ExecStmt' 'Int32[] dateNums;'
⎕cse 'SetValue' 'dateNums' (20160101 20160513 20161231)
```

```
ρ□←⊃□cse 'GetValue' 'DayDiffs({0},dateNums)' 20160101
□cse 'Close'
```

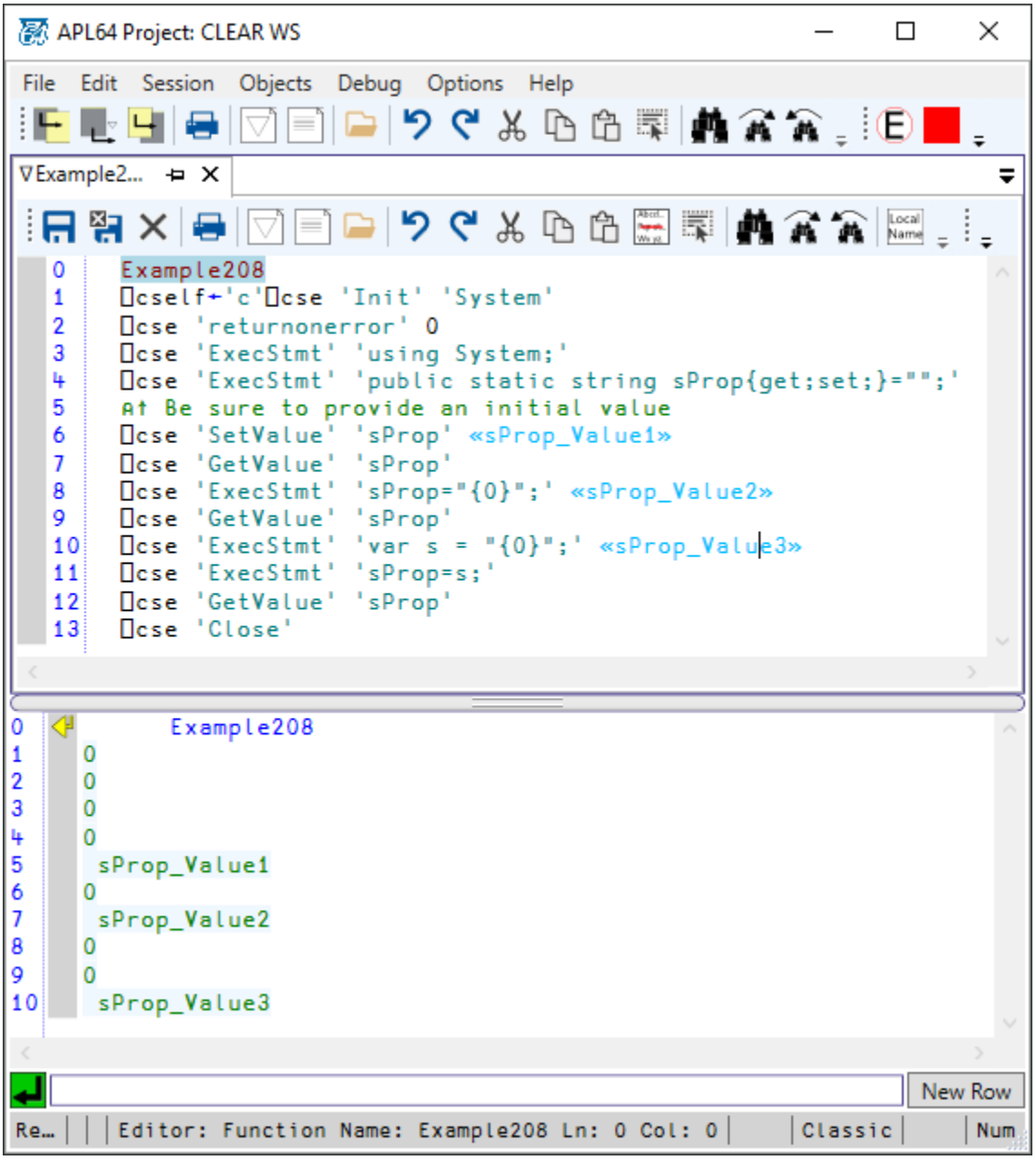


Example #208 SetValue for Public static string property

In this example a public static property of string type is defined and its value set in three ways.

```
Example208
□cself←'c'□cse 'Init' 'System'
□cse 'returnonerror' 0
□cse 'ExecStmt' 'using System;'
□cse 'ExecStmt' 'public static string sProp{get;set;}="";'
⍝↑ Be sure to provide an initial value
□cse 'SetValue' 'sProp' «sProp_Value1»
□cse 'GetValue' 'sProp'
□cse 'ExecStmt' 'sProp="{0}";' «sProp_Value2»
□cse 'GetValue' 'sProp'
□cse 'ExecStmt' 'var s = "{0}";' «sProp_Value3»
□cse 'ExecStmt' 'sProp=s;'
```

```
cse 'GetValue' 'sProp'
cse 'Close'
```



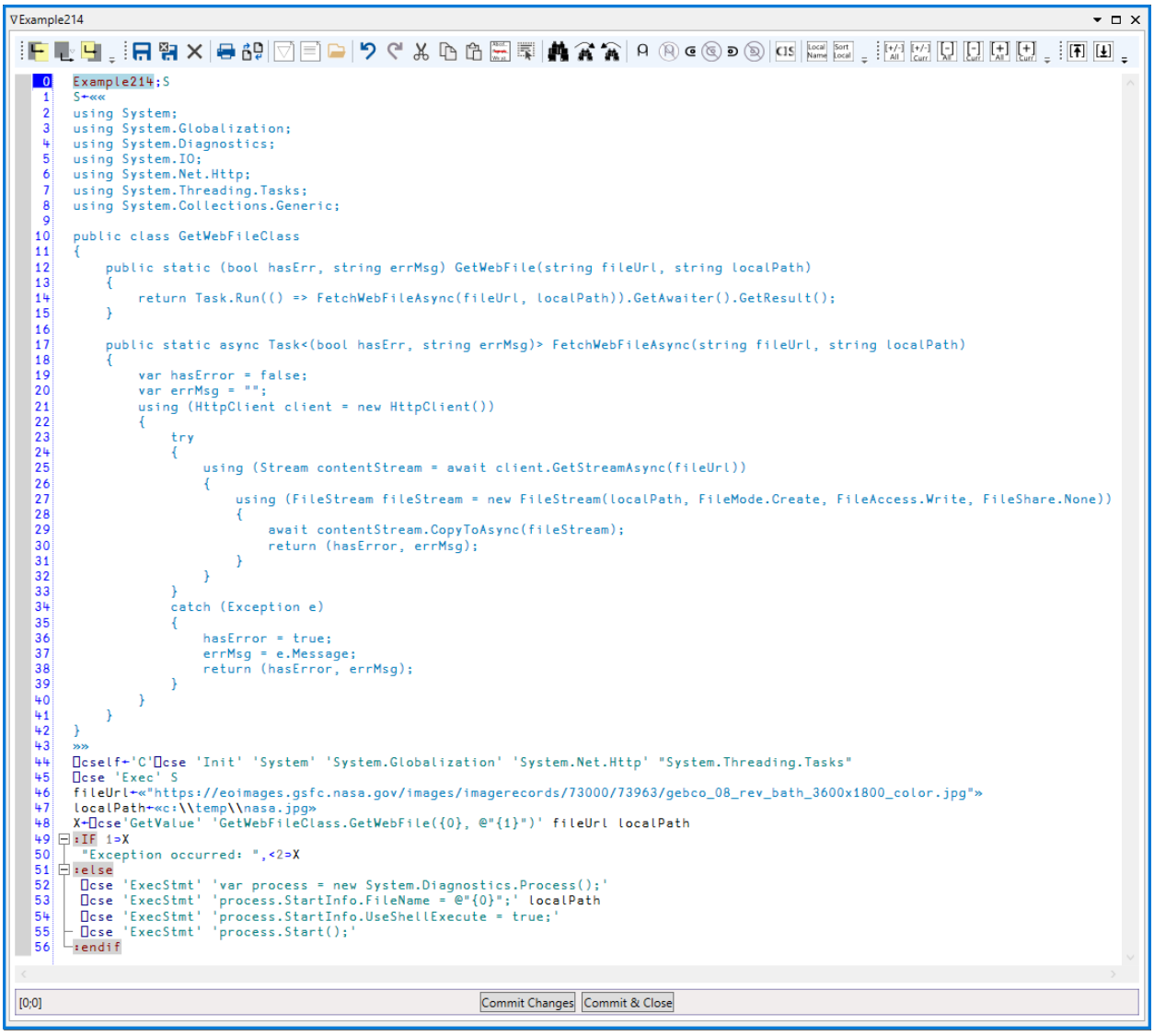
Example #214: Use .Net to download file via http or https

This example uses the .Net HttpClient class to download a file from a web url.

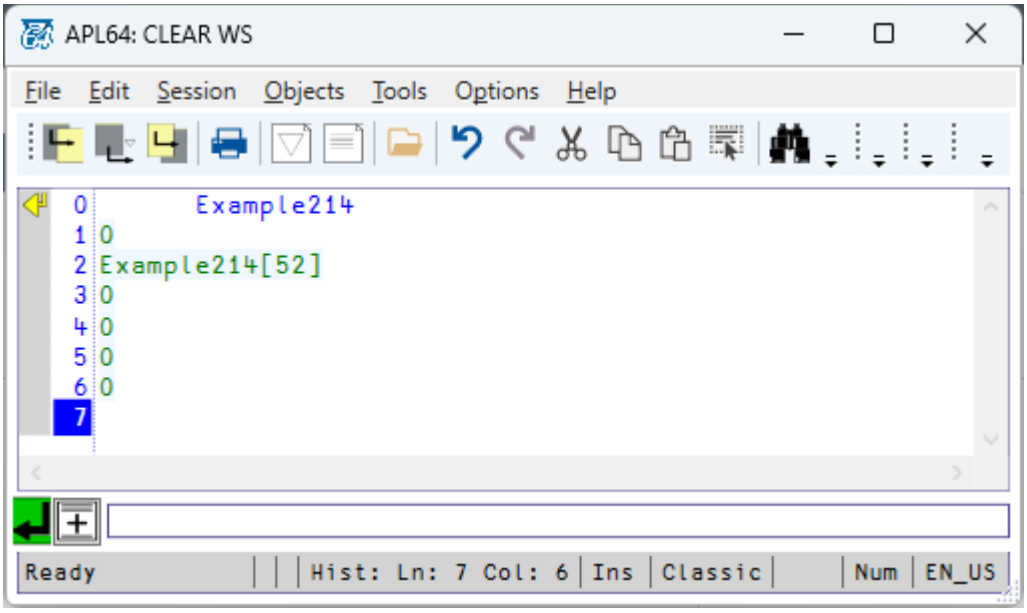
Create the Example214 function using this definition:

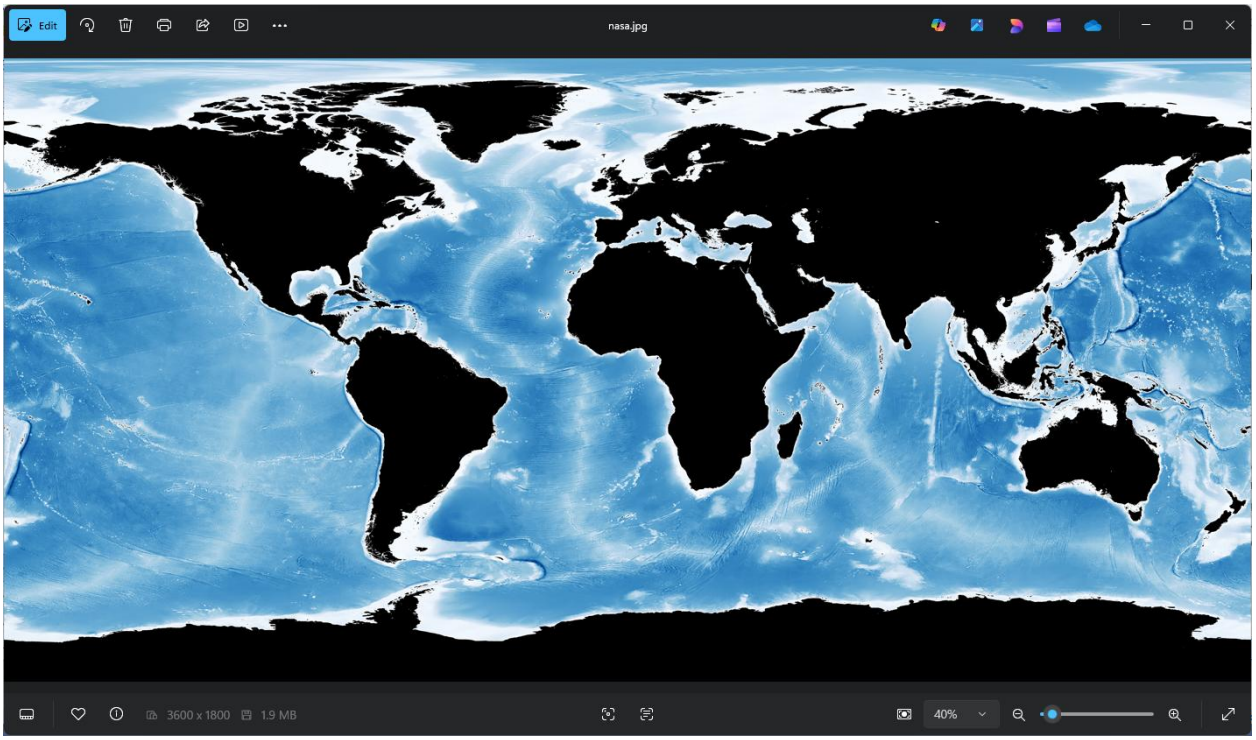
```
Example214;S
S←««
using System;
using System.Globalization;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Collections.Generic;

public class GetWebFileClass
{
   public static (bool hasErr, string errMsg) GetWebFile(string fileUrl, string localPath)
   {
      return Task.Run(() => FetchWebFileAsync(fileUrl, localPath)).GetAwaiter().GetResult();
   }

   public static async Task<(bool hasErr, string errMsg)> FetchWebFileAsync(string fileUrl, string localPath)
   {
      var hasError = false;
      var errMsg = "";
      using (HttpClient client = new HttpClient())
      {
         try
         {
            using (Stream contentStream = await client.GetStreamAsync(fileUrl))
            {
               using (FileStream fileStream = new FileStream(localPath, FileMode.Create, FileAccess.Write, FileShare.None))
               {
                  await contentStream.CopyToAsync(fileStream);
                  return (hasError, errMsg);
               }
            }
         }
         catch (Exception e)
         {
            hasError = true;
            errMsg = e.Message;
            return (hasError, errMsg);
         }
      }
   }
}
»»
□cself←'C'□cse 'Init' 'System' 'System.Globalization' 'System.Net.Http' "System.Threading.Tasks"
```

```
⎕cse 'Exec' S
fileUrl←«"https://eoimages.gsfc.nasa.gov/images/imagerecords/73000/73963/gebco_08_rev_bath_3600x1800_color.jpg"»
localPath←«c:\\temp\\nasa.jpg»
X←⎕cse'GetValue' 'GetWebFileClass.GetWebFile({0}, @"{1}")' fileUrl localPath
:IF 1⊃X
 "Exception occurred: ",<2⊃X
:else
 ⎕cse 'ExecStmt' 'var process = new System.Diagnostics.Process();'
 ⎕cse 'ExecStmt' 'process.StartInfo.FileName = @"{0}";' localPath
 ⎕cse 'ExecStmt' 'process.StartInfo.UseShellExecute = true;'
 ⎕cse 'ExecStmt' 'process.Start();'
:endif
```



When this example runs, the Windows operating system, invoked by the Process.Start() method, will display the image file using the default image display program select for the target workstation



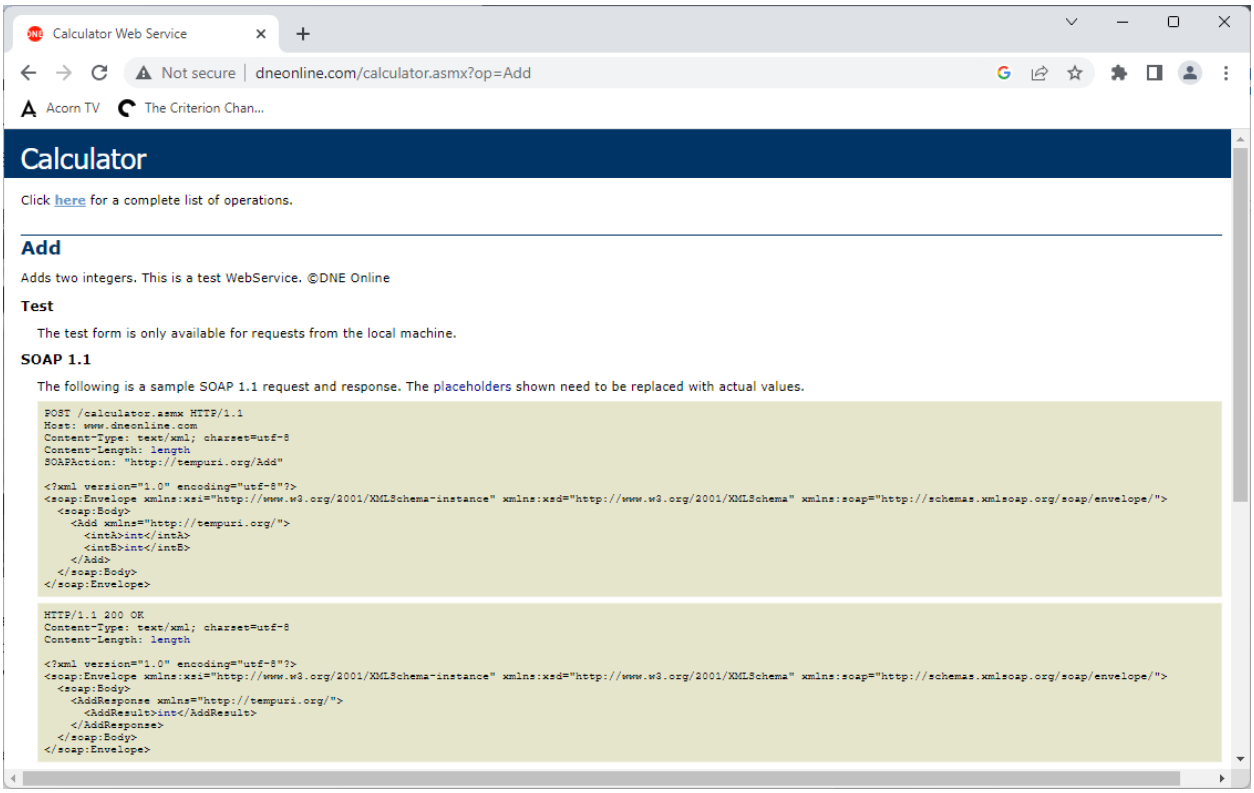


Example #247: Accessing a Soap Web Service

Soap is a methodology for exposing a web service to a client. In this example the CSE is used to access a sample Soap web service. To access a Soap web service, the xml format of its soap request and soap response must be known. Generally, this is available by reviewing the web service's wsdl.

The example soap web service url: http://www.dneonline.com/calculator.asmx

The wsdl for the sample web service in this example can be obtained using an Internet browser:

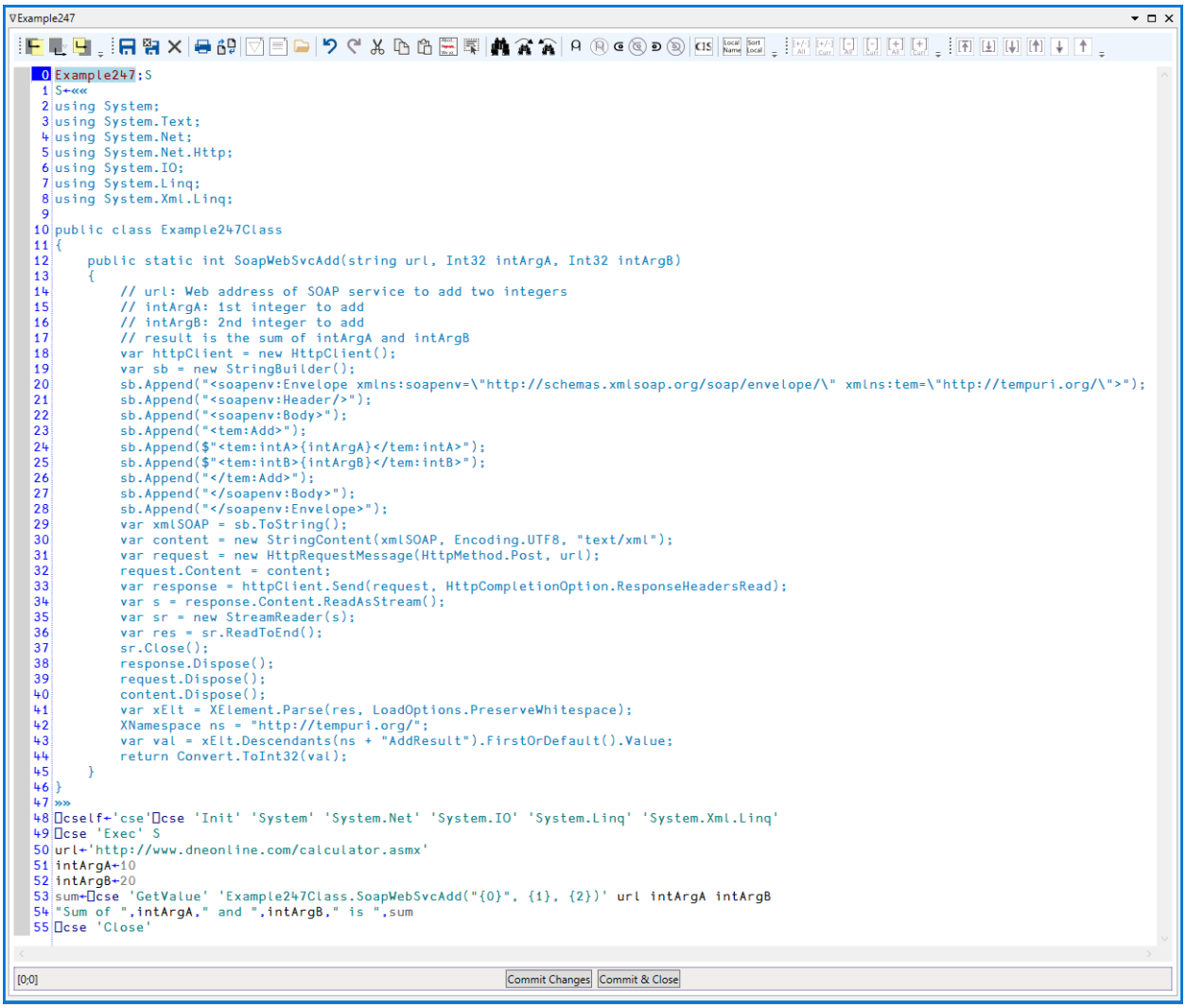


APL64 function to use the sample Soap web service to add two integers, using the web-based calculator:

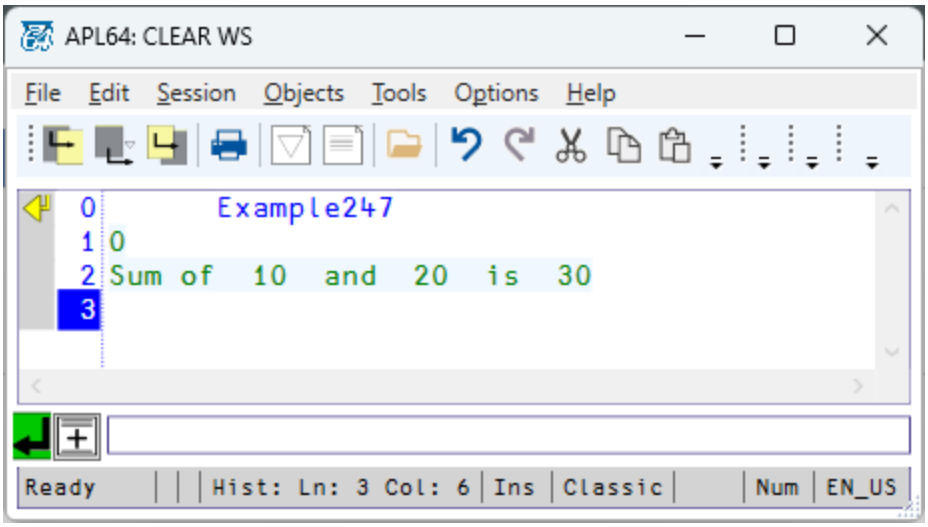
```
Example247;S
S←««
using System;
using System.Text;
using System.Net;
using System.Net.Http;
using System.IO;
using System.Linq;
using System.Xml.Linq;

public class Example247Class
{
```

```
   public static int SoapWebSvcAdd(string url, Int32 intArgA, Int32 intArgB)
   {
      // url: Web address of SOAP service to add two integers
      // intArgA: 1st integer to add
      // intArgB: 2nd integer to add
      // result is the sum of intArgA and intArgB
      var httpClient = new HttpClient();
      var sb = new StringBuilder();
      sb.Append("<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:tem=\"http://tempuri.org/\">");
      sb.Append("<soapenv:Header/>");
      sb.Append("<soapenv:Body>");
      sb.Append("<tem:Add>");
      sb.Append($"<tem:intA>{intArgA}</tem:intA>");
      sb.Append($"<tem:intB>{intArgB}</tem:intB>");
      sb.Append("</tem:Add>");
      sb.Append("</soapenv:Body>");
      sb.Append("</soapenv:Envelope>");
      var xmlSOAP = sb.ToString();
      var content = new StringContent(xmlSOAP, Encoding.UTF8, "text/xml");
      var request = new HttpRequestMessage(HttpMethod.Post, url);
      request.Content = content;
      var response = httpClient.Send(request, HttpCompletionOption.ResponseHeadersRead);
      var s = response.Content.ReadAsStream();
      var sr = new StreamReader(s);
      var res = sr.ReadToEnd();
      sr.Close();
      response.Dispose();
      request.Dispose();
      content.Dispose();
      var xElt = XElement.Parse(res, LoadOptions.PreserveWhitespace);
      XNamespace ns = "http://tempuri.org/";
      var val = xElt.Descendants(ns + "AddResult").FirstOrDefault().Value;
      return Convert.ToInt32(val);
   }
}
»»
□cself←'cse'□cse 'Init' 'System' 'System.Net' 'System.IO' 'System.Linq' 'System.Xml.Linq'
□cse 'Exec' S
url←'http://www.dneonline.com/calculator.asmx'
intArgA←10
intArgB←20
```

```
sum←⎕cse 'GetValue' 'Example247Class.SoapWebSvcAdd("{0}", {1}, {2})' url intArgA intArgB
"Sum of ",intArgA," and ",intArgB," is ",sum
⎕cse 'Close'
```



Run the Example247 function:



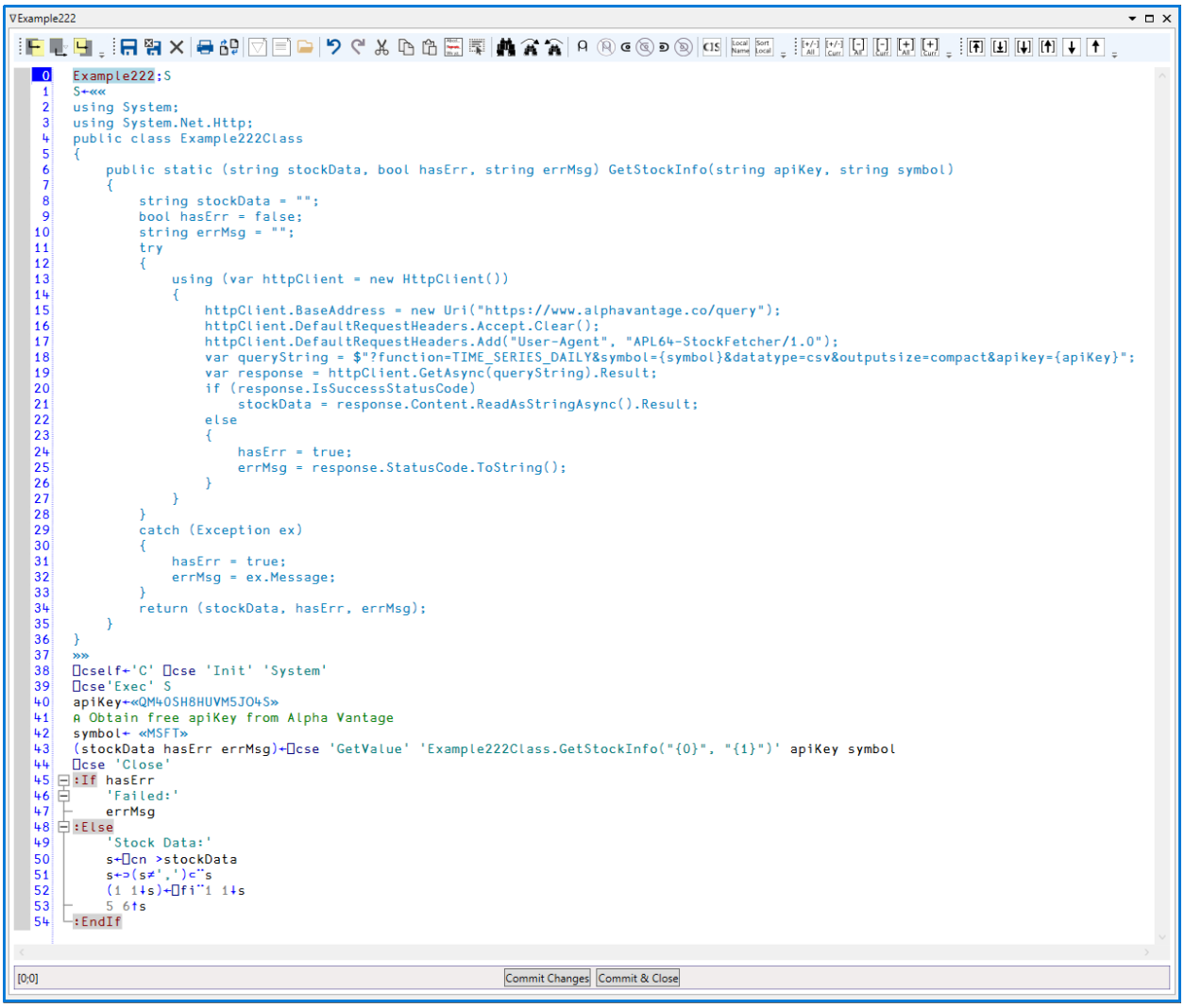
Obtaining Financial Stock Information
Example222

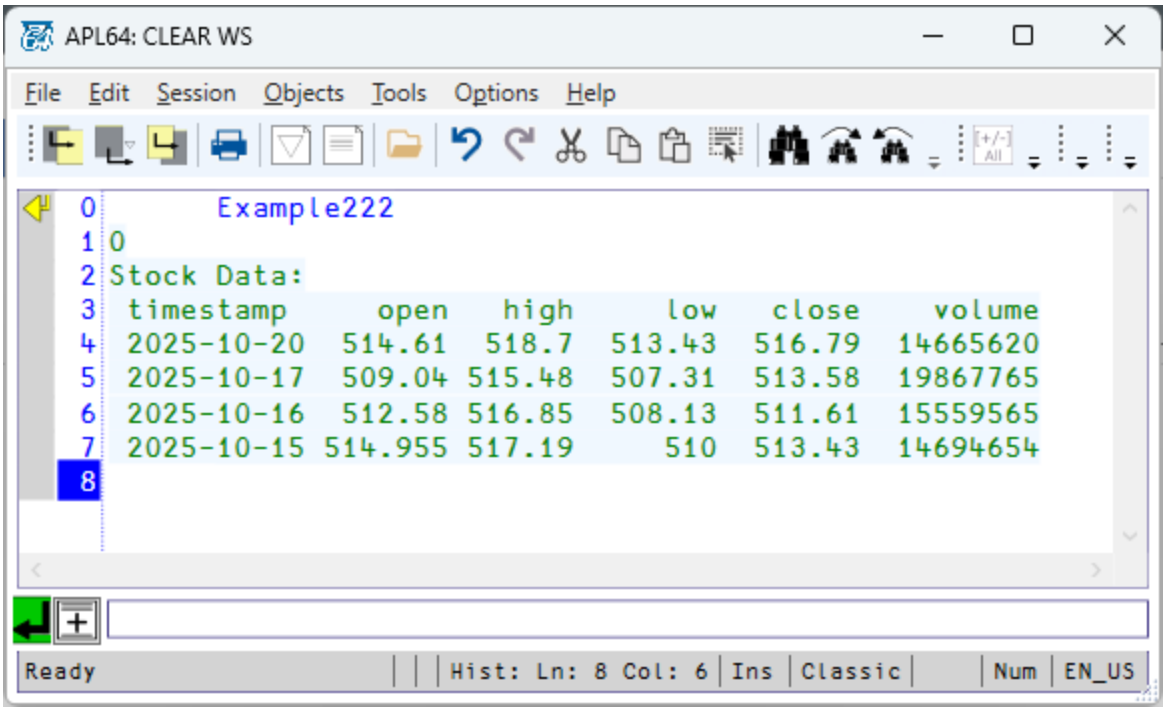
In this example a financial company has made available a Web API which can be accessed as a Web API. To use the Web API, a free API key must be obtained, in exchange for an email address. APL2000 has no affiliation with 'AlphaVantage'. Some stock symbols: MSFT, GOOGL, BRK.A, PFE, TSM.

Create the Example222 function using this definition:

```
Example222;S
S←««
using System;
using System.Net.Http;
public class Example222Class
{
   public static (string stockData, bool hasErr, string errMsg) GetStockInfo(string apiKey, string symbol)
   {
      string stockData = "";
      bool hasErr = false;
      string errMsg = "";
      try
      {
         using (var httpClient = new HttpClient())
         {
            httpClient.BaseAddress = new Uri("https://www.alphavantage.co/query");
            httpClient.DefaultRequestHeaders.Accept.Clear();
            httpClient.DefaultRequestHeaders.Add("User-Agent", "APL64-StockFetcher/1.0");
```

```
            var                                                queryString                                                =
$"?function=TIME_SERIES_DAILY&symbol={symbol}&datatype=csv&outputsize=compact&apikey={api
Key}";
            var response = httpClient.GetAsync(queryString).Result;
            if (response.IsSuccessStatusCode)
               stockData = response.Content.ReadAsStringAsync().Result;
            else
            {
               hasErr = true;
               errMsg = response.StatusCode.ToString();
            }
         }
      }
      catch (Exception ex)
      {
         hasErr = true;
         errMsg = ex.Message;
      }
      return (stockData, hasErr, errMsg);
   }
}
»»
□cself←'C' □cse 'Init' 'System'
□cse'Exec' S
apiKey←«QM40SH8HUVM5JO4S»
⍝ Obtain free apiKey from Alpha Vantage
symbol← «MSFT»
(stockData hasErr errMsg)←□cse 'GetValue' 'Example222Class.GetStockInfo("{0}", "{1}")' apiKey
symbol
□cse 'Close'
:If hasErr
   'Failed:'
   errMsg
:Else
   'Stock Data:'
   s←□cn >stockData
   s←⊃(s≠',')⊂¨s
   (1 1↓s)←□fi¨1 1↓s
   5 6↑s
:EndIf
```


VExample222
Example222:S
S←««
using System;
using System.Net.Http;
public class Example222Class
{
public static (string stockData, bool hasErr, string errMsg) GetStockInfo(string apiKey, string symbol)
{
string stockData = "";
bool hasErr = false;
string errMsg = "";
try
{
using (var httpClient = new HttpClient())
{
httpClient.BaseAddress = new Uri("https://www.alphavantage.co/query");
httpClient.DefaultRequestHeaders.Accept.Clear();
httpClient.DefaultRequestHeaders.Add("User-Agent", "APL64-StockFetcher/1.0");
var queryString = $"?function=TIME_SERIES_DAILY&symbol={symbol}&datatype=csv&outputsize=compact&apikey={apiKey}";
var response = httpClient.GetAsync(queryString).Result;
if (response.IsSuccessStatusCode)
stockData = response.Content.ReadAsStringAsync().Result;
else
{
hasErr = true;
errMsg = response.StatusCode.ToString();
}
}
}
catch (Exception ex)
{
hasErr = true;
errMsg = ex.Message;
}
return (stockData, hasErr, errMsg);
}
}
»»
⎕cself←'C' ⎕cse 'Init' 'System'
⎕cse'Exec' S
apiKey←«QM40SH8HUVM5JO4S»
⍝ Obtain free apiKey from Alpha Vantage
symbol← «MSFT»
(stockData hasErr errMsg)←⎕cse 'GetValue' 'Example222Class.GetStockInfo("{0}", "{1}")' apiKey symbol
⎕cse 'Close'
:If hasErr
'Failed:'
errMsg
:Else
'Stock Data:'
s←⎕cn ⍬stockData
s←⍉(s≠',')⌿⍤1⊢s
(1 1↓s)←⎕fi⍤1 1↓s
5 6↑s
:EndIf
[0;0]
Commit Changes
Commit & Close



APL+Win CSE to APL64 CSE

Generally APL+Win CSE applications need few, if any, modifications to be used with the APL64 CSE. The APL64 CSE incorporates many improvements which were not possible to implement in the APL+Win CSE including:

- Use the APL64 CSE manual for APL64, and not the APL+Win CSE
- All examples with source code are in the APL64 CSE manual. A separate APL64 CSE example (.zip) file is not necessary.
- The APL64 CSE has no dependency on Windows-only ActiveX technology
- APL64 CSE object and instances are now in-process features of APL64
    - No separate APL64 CSE installer required.
    - Performance is improved because there is no APL64/CSE boundary requiring data serialization.
    - Faster APL64 CSE initialization
    - No SignalR server technology needed
    - No separate application domain for the CSE engine needed.
    - APL64 CSE instance InitSTA method now operates the same as the CSE instance Init method.
    - APL64 CSE logging is performed directly in the APL64 interpreter instance.
- The APL64 CSE is cross-platform compatible.
- The APL64 CSE requires no 'port numbers', and they are needed in the right argument of the APL64 CSE Init and InitSTA methods.
- The Exec method C# script argument may be a character vector, character matrix or scalar string.

- The ExecFile method no longer requires the Boolean argument indicating if the filed script contains APL64 executable statements prefixed by 'APL:'
- The CSE method and property keywords are not case-sensitive, e.g. 'Exec', 'EXEC', 'trackevents', 'TrackEvents' are acceptable. C# keywords remain case sensitive.
- The □cself system variable property is no longer initialized when the workspace is cleared or a workspace is loaded
- CSE Close method optional argument no longer applicable and will be ignored if present.
- CSE instance method, GetValueEx, is available so that the desired result can be distinguished from an error code, so that the value of the APL64 programmer selected CSE instance property, ReturnOnError, can be considered for this method.
- The CSE instance method GetLastError now has an optional right argument indicating the number of lines of the last error which will be returned by this method.
- CSE instance property RepStr value must be explicitly set by the APL64 programmer.
- APL+Win CSE instance properties deprecated:
    - texttransfer: APL64 supports the string data type. Use an APL64 string datatype to set a C# string datatype and an APL64 char datatype to set a C# char datatype.
    - transpose: APL64 CSE automatically handles the row and column order differences between APL and .Net.
    - unicode: APL64 directly supports Unicode
    - logfilepath: Since the APL64 CSE is an in-process engine, logging is performed by the APL64 interpreter.
- APL+Win CSE Object properties deprecated:
    - ReconnectionTimeout: APL64 CSE is an in-process engine, so no server technology is required.
    - ConnectionTimeout: APL64 CSE is an in-process engine, so no server technology is required.
    - ConnectionInterval: APL64 CSE is an in-process engine, so no server technology is required.
    - TraceLevel: APL64 CSE is an in-process engine, so event tracing is performed by the APL64 interpreter.
    - Version: APL64 CSE is an in-process engine, so the CSE engine version is the same as the APL64 interpreter version.
- For the CSE instance methods, ExecStmt, GetValue, GetValueEx, SetValue methods, the target .Net variable data type more closely determines the associated APL64 data type.
    - APL64 bool values (0/1) will be converted to Int32 values when sent to the CSE engine. APL64 Boolean values or Int32 (0/1) values may be sent to the server to set the value of CSE objects.
    - .Net bool values will be converted to APL64 Int32 values or bool values via demotion, if applicable.
    - .Net string values will be returned to APL64 as string values. To send string values to a .Net string variable, use an APL64 string value
    - .Net char values will be returned to APL64 as character values. To send char values to a .Net char variable, use an APL64 character type.
    - Individual elements of a .Net Object array containing non-APL64 data types must be obtained using the CSE GetValue method, rather than obtaining the entire object array via GetValue.

- APL64 and the CSE in APL64 are implemented using .Net and the Microsoft Roslyn C# compiler to achieve cross-platform compatibility. In this environment, the CSE cannot be used to subscribe to events which are exposed by System.Windows.Forms or Windows Presentation Foundation (WPF) GUI controls. The CSE in APL+Win was implemented on the .Net Framework which is not cross-platform compatible.