Session Commands (Visual APL)

The Cielo Explorer includes a wide range of commands for managing the various aspects of the session.

These aspects include the listing of session contents, script management, and editing of variables.

)cd

Changes the current context of the session into or out of the specified object.

Syntax:

```
)cd obj
```

obj: The name of a class, variable, or path control string.

Remarks:

This session command provides the ability to explore classes. It is possible to explore either an instance of a class or the class itself.

When this session command is used without an argument it displays the current class being explored, for instance when at the top level, in the session, this is displayed

```
      )cd
session
```

The right argument to the)cd session command is either a classname, a variable or a relative path.

To navigate to a particular class:

```
      )cd classname
```

To navigate to an instance of a class:

```
      a = classname()
      )cd a
```

To navigate to a relative location:

```
      )cd ../../a/b/c
```

Or to navigate up one level:

```
      )cd ..
```

To return to the session or root:

```
      )cd /
```

Examples and narrative:

Once you have navigated into a class, you will see all of the methods, properties, events and fields in the class, regardless of member attributes. This means you can review members which are public, internal, private, etc.

As an example, consider an integer:

```
  a = 10
  )cd a
Loaded instance of: System.Int32
```

Now if we look at the)fns in this instance of the ValueType Int32 we see:

```
  )fns
_dataRepresentation CompareTo           Equals              Finalize
           GetHashCode
GetType             GetTypeCode         MemberwiseClone     Parse
              ToString
TryParse
```

The)fns includes methods, functions and the methods associated with properties.

Now if we look at)vars we see:

```
     )vars

m_value  MaxValue MinValue
```

We can navigate back up to the session by entering)cd ..

```
     )cd ..
Loaded instance of: APLNext.APL.Objects.module
```

If we try to navigate up again:

```
     )cd ..
Current instance is session
```

We see that we are already at the session level, and cannot navigate further up.

While we are back at the session level, let's consider what is visible on the Int32 we have placed in the variable a.

If we look at intellisense on an instance of an Int32, we see a small subset of those members we saw when we navigated into the instance of Int32 on the variable a.

Specifically, if we navigate back into the a variable:

```
     )cd a
Loaded instance of: System.Int32
```

If we then check the variable m_value, which is not normally available to investigate, we find:

```
     m_value
10
```

So we see that the Int32 is an object, a ValueType in particular, and that the integer value is stored on the field m_value.

If we want to know where we are in our navigation, we can always do)cd without an argument:

```
     )cd
session/a
```

No matter how deep we have navigated we can always move back to the session by entering:

```
    )cd /
Current instance is session
```

To review, everything is an object, and we can navigate through those objects using)cd, in this example lets look at an Int32 and navigate down and up through this object.

```
     a = 10
     )cd a
Loaded instance of: System.Int32

      )vars
m_value  MaxValue MinValue
```

```
      m_value
10

      )cd MaxValue
Loaded instance of: System.Int32

      )vars
m_value  MaxValue MinValue

      m_value
2147483647

      )cd
session/a/MaxValue

      )cd ..
Loaded instance of: System.Int32

      )cd
session/a

      )cd ..
Loaded instance of: APLNext.APL.Objects.module

      )cd
session
```

)classes

Shows the current list of classes which have been defined in the session.

Syntax:

```
)classes
```

Remarks:

The *)classes* command shows the list of classes which have been created in the session.

Classes are most commonly created in the session by running a script file.

Example:

Here is an example script which contains the definition of two classes:

```
// Script: sc1

public class math {
    function add(a, b) {
        return a + b
    }

    function subtract(a, b) {
        return a - b
    }
}

public class useMath {
    function fn(a, b) {
        m = math()
        return m.add(a, b)
    }
}
```

Now lets run the script to create the classes in the session:

```
      // display the contents of the )classes list
      )classes

      // the list is empty

      // load and run the script 'sc1'
      )load sc1

      // display )classes again
      )classes
math usemath
      // the two classes now exist in the session.

      // run the useMath class:
      um = useMath()
      um.fn(10, 20)
30
```

)clear

Clears all variables, functions, etc, from the active Cielo Explorer.

Syntax:

```
)clear
```

Remarks:

When the clear session command is run in the Cielo Explorer, the .Net AppDomain which currently represents the Cielo Explorer is shutdown, thus removing from memory any variables, functions, UDF's, file ties, etc, from memory.

If an assembly is referenced into the session by the use of the refbyname or refbyfile directives, then the DLL which that reference represents is tied in memory by the session. This behavior is required by the .Net security model.

When the Cielo Explorer AppDomain is unloaded by the clear command, any assembly references are also removed from memory, meaning that any tied assemblies can again be modified, recreated, and moved on the disc.

Example:

```
      a = 10 20 30
      a
10 20 30

      )clear

      a
The variable 'a' does not exist
```

)edit

Opens a script file for editing in the current Session Project of the active Solution.

Syntax:

```
)edit script
```

script: The name of the script file to create or open.

Remarks:

When the edit command is run in the Cielo Explorer session, a script file of the specified name is opened for editing from the current Session Project in Visual Studio, and given the active window focus.

If the script file does not exist in the current Session Project, then a new script file is created in the Session Project by the supplied name, and opened for editing.

If the script file does exist in the current Session Project, then it is opened and focused for editing.

If the script file is already open in Visual Studio, then that script file is brought to the forefront and receives the window focus.

Saving a script

Once you have edited the contents of a script file, you can save and execute the script to the Cielo Explorer session by pressing the key sequence **Ctrl+E+E**. Once the script is saved and executed to the Cielo Explorer, a message is printed to the session stating that the script was modified and imported.

Example:

```
      // make a variable in the session
      a = 10 20 30
      a
10 20 30

      // edit a new script
      )edit sc

      // add this line to the script
      a = a + 100

      // save the script by pressing Ctrl+E+E.

      // this line is printed to the session
      Script 'sc' updated

      // now again display the contents of 'a' in the session.
      a
110 120 130
```

)fns

Shows the current list of functions which have been defined in the session.

Syntax:

```
)fns
```

Remarks:

The *)fns* command shows the list of functions which have been created in the session.

Here are a few examples of creating functions in the session:

- Entering its declaration directly in the session:

```
      // display the contents of the )fns list
      )fns

      // the list is empty

      // define a new function called 'add'
      function add(a, b) { return a + b }

      // display )fns again
      )fns
add
      // the function 'add' is now in the list.
```

- Using def to declare a function from a text string:

```
      // display the contents of the )fns list
      )fns

      // the list is empty

      // define a function from a string
      def "function add(a, b) { return a + b }"
true

      // display )fns again
      )fns
add
      // the function 'add' is now in the list.
```

- Execute a script which contains the definition of one or more functions. Here is an example script which contains the definition of two functions:

```
// Script: sc1

function add(a, b) {
      return a + b
}
```

```
function subtract(a, b) {
      return a - b
}
```

Now lets run the script to create the functions in the session:

```
      // display the contents of the )fns list
      )fns

      // the list is empty

      // load and run the script 'sc1'
      )load sc1

      // display )fns again
      )fns
add subtract
      // the functions are now in the list.
```

These are only a few simple examples of creating functions in the session. Any valid expression or statement which creates a function can be run is the session, and once that command is run the resultant function will be present in the *)fns* list.

Created Function Time Stamping

When functions are dynamically created in the session, they receive an associated DateTime object which represents the moment that the function was created in the session. There are several system quad functions which allow the retrieval and modification of this time stamp.

Example:

```
      // check the contents of the )fns list
      )fns
mult sub
      // there are two functions currently defined

      // define a new function called 'add'
      function add(a, b) { return a + b }

      // check the )fns list
      )fns
add mult sub
      // the function 'add' is now in the list.

      // try running add:
      10 add 20
30
```

)load

Loads and executes a script from the current Session Project.

Syntax:

```
)load script
```

script: The name of the script file to load.

Remarks:

The)load command looks in the current Session Project for a script named the specified name. If the script file exists, it is added to the)scripts list in the session, and then the contents of the script are executed in the session.

This command has the same behavior as pressing **Ctrl+E+E** in an open script in Visual Studio.

Example:

```
      // check if 'a' exists
      a
name 'a' is not defined

      // check the contents of the )script list
      )scripts

      // the )scripts list is empty

      // load a script which defines 'a'
      )load sc

      // check if 'a' exists
      a
10 20 30

      // check the )scripts list
      )scripts
sc
      // the script 'sc' is now in the list.
```

)off

Clears all variables, functions, etc, from the active Cielo Explorer. Also closes the current open Solution in Visual Studio.

Syntax:

```
)off
```

Remarks:

The *off* command has the same effect as the *clear* command for the contents of the session, and also closes the currently open Solution in Visual Studio, and all associated projects.

If any open files are currently marked as unsaved in Visual Studio, then the Save File dialog is opened prompting for user action. This behavior is the built-in functionallity of Visual Studio, meaning that in relation to the currently open Solution, the off command has the same effect as the "File > Close Solution" menu item.

Example:

```
      a = 10 20 30
      a
10 20 30

      )off

      a
The variable 'a' does not exist
```

)run

Executes the contents of a script from the current Session Project.

Syntax:

```
)run script
```

script: The name of the script file to run.

Remarks:

The *)run* command looks in the *)scripts* list for a script named the specified name. If the script file exists in the list, the contents of the script are executed in the session.

The *)run* command is similar to the *)load* command, except that the specified script must already be listed in the *)scripts* list for the command to succeed.

Example:

```
      // check if 'a' exists
      a
name 'a' is not defined

      // check the contents of the )script list
      )scripts
sc
      // the 'sc' script is present in the session

      // run the script 'sc', which defines 'a'
      )run sc

      // check if 'a' exists
      a
10 20 30
```

)runf

Executes the contents of a script at the specified file path.

Syntax:

```
)runf scriptPath
```

scriptPath: The fully qualified file path of the script to run.

Remarks:

The *)runf* command takes a file path to a script file as its argument. When the *)runf* command is entered, the contents of the specified script file are run in the session.

The *)runf* command is similar to the *)run* command, except that the argument script file does not need to exist in the *)scripts* list.

Example:

```
      // check if 'a' exists
      a
name 'a' is not defined

      // check the contents of the )script list
      )scripts

      // the list is empty

      // runf the script 'sc', which defines 'a'
      )runf c:\sc.apl

      // check if 'a' exists
      a
10 20 30
```

)scripts

Shows a list of scripts which are currently loaded into the session.

Syntax:

```
)scripts
```

Remarks:

Script files can be loaded into the session by the *)load*, *)xload*, and *)edit* commands. Pressing **Ctrl+E+E** in an open script file also adds that script to the *)scripts* list.

Example:

```
      // check the contents of the )script list
      )scripts
sc1 sc2 sc3
      // there are three scripts currently loaded

      // xload a script called 'math'
      )xload math

      // check the )scripts list
      )scripts
sc1 sc2 sc3 math
      // the script 'math' is now in the list.
```

)vars

Shows the current list of variables which have been defined in the session.

Syntax:

```
)vars
```

Remarks:

The *)vars* command shows the list of variables which have been created in the session.

Example:

```
      // check the contents of the )vars list
      )vars
a b
      // there are two variables currently defined

      // define a new variable called 'c'
      c = 100 200 300

      // check the )vars list
      )vars
a b c
      // the variable 'c' now exists in the session.
```

)xload

Loads a script from the current Session Project.

Syntax:

```
)xload script
```

script: The name of the script file to xload.

Remarks:

The *)xload* command looks in the current Session Project for a script named the specified name. If the script file exists, it is added to the *)scripts* list in the session.

This command has a similar behavior to the *)load* command, except that the contents of the script are not executed.

Example:

```
      // check the contents of the )script list
      )scripts

      // the )scripts list is empty

      // xload a script called 'sc'
      )xload sc

      // check the )scripts list
      )scripts
sc
      // the script 'sc' is now in the list.
```

)xmlout

Exports a variable in XML format into the current Session Project in Visual Studio.

Syntax:

```
)xmlout var var var...
```

var: The name of a variable to export.

Remarks:

For each argument variable, the *xmlout* command creates an XML file in the current Session Project named "*var*.xml", where *var* is the name of the variable being exported.

If an XML file by that name already exists in the current Session Project, then that file is overwritten with the newly produced XML output.

Only objects which are serializable can be successfully exported to XML.

A variable is considered serializable if any of the following conditions are met:

- It is marked with the .Net Serializable attribute.
- Implements the .Net ISerializable interface.
- Implements the IXMLSerializable interface.
- Has a registered serializer in the Cielo Explorer.

If an object is encountered in a variable being exported with *xmlout* that does not meet any of the above serializable criteria, then a comment is placed in the generated XML at the location where the object would have appeared in the output XML, stating that the element could not be serialized. This behavior ensures that if you have a variable in the Cielo Explorer which contains mostly serializable data and only a few elements which cannot be serialized, the elements which are not serialzable will not prevent the serializable elements from being exported to XML format. Keep in mind that if you save the generated XML back to the Cielo Explorer by the use of **Ctrl+E+E**, that those elements which could not be serialized during the generation of the XML file will contain empty objects in the newly imported variable.

Once the *xmlout* command has been executed, the active window in Visual Studio is shifted to the newly created or updated XML file. If more than one variable was exported, the focus is placed on each XML file view as it is created, ultimately being placed on the XML file of the last variable being exported.

Saving changes to XML variables

Once a variable has been exported as XML, the generated XML can be modified in any way desired, and those changes can be saved back to the Cielo Explorer.

To save changes made to an XML file in the current Session Project, open that file and press the key sequence **Ctrl+E+E**

Once the sequence is pressed, focus is returned to the Cielo Explorer, and a message is printed to the session showing that an update was made to the variable.

When the changed XML is saved back to the session, the name of the variable into which the data is saved is taken from the name of the XML file. This means that if an XML file named "a.xml" is saved to the session using **Ctrl+E+E**, then the deserialized contents of that file will be saved as the variable "a" in the session. This means that you can not only save variables from the session in XML format, but you can also import entirely new variables from XML format by simply adding them to the Session Project, opening them, and

then pressing **Ctrl+E+E**

Additional Information

The *xmlout* command uses the XmlCvarSerialzier to perform the XML conversion to and from the current Session Project.

Example:

```
a = 10 "test" (⍳5)
)xmlout a
// a file has been created in the current Session Project.
```

Cielo Explorer Menu Reference

The Cielo Explorer includes a toolbar which allows various common session management activities to be easily performed at the click of a button.

Following is a listing of each button in the Cielo Explorer and their uses:

Toolbar buttons in Cielo Explorer

New

Clears the present session. This unloads the present domain, removing all references to assemblies and creates a new session domain.

Run Cielo Script

The user is prompted with the file selection dialog box. A script file is selected which is then defined and run in the current session. Scripts can contain any statement or expression; this includes control structures, function definitions, classes and simple statements. Scripts are dynamic, and functions, variables or classes defined in a script replace any dynamic members that exist in the current session with the same names.

For instance, the following script defines the function fn and then calls that function:

```
// Script: sc
function fn(a) {
           return a
}
fn( hello )
```

When this script is run in the session, the word hello is displayed in the session and the function fn is added to those available in the session.

Load Cielo File

This loads a Visual APL file which contains a formal assembly definition. The assembly is created and the resulting dll or exe can be referenced in the session or in the case of an exe, run from the OS.

Import Assembly

This adds a reference to an existing assembly to the session. If there is a namespace in the assembly which matches the name of the assembly, a using is also done.

Load Session Log

This prompts the user with a file selection dialog box. The user can choose any existing Visual APL log file, which is then displayed in the session, thus providing the user with all of the commands, definitions, etc which occurred in a previous session.

Save Session Log

This action saves all of the display content in the existing session to the file selected by the user. The user is prompted with the Save File dialog box.

Print

This prints the display contents of the existing session.

Cut

This removes the selected text from the session display and places it on the clipboard.

Copy

This copies the selected text which is placed on the clipboard.

Paste APL+Win

This pastes APL+Win code into the session explicitly converting from the legacy APL+Win text to APL Unicode.