Changes to SALT/UCMDs from V13 to V13.1

General changes

V11 is no longer supported.

The class ⎕SE.UnicodeFile is no longer used by SALT but is still available for general use.

SALT has always been residing in the SALT folder under the installation path. In some organizations it may be preferred to use a central location instead of individual copies. You can now specify where the SALT folder resides if you wish. This is done by adding an environment variable named SALT. This can be achieved on the command line.

Under Unix the syntax is

SALT=/path/… dyalog …

Under Windows it is

dyalog.exe … SALT=\path\…

The workspace BUILDSE no longer has a separate copy of some of the SALT code but rather relies on the *SALTUtils* namespace, which contains all the bootstrapping code, to generate SALT and Spice (the UCMD framework).

The structure of the folders has changed:

- The folder SALT\SALT is now named SALT\core
- The subfolders under Tools have been renamed and reassigned

All SALT commands now accept omega (ω) in the 1st position to mean "same folder as the current workspace". Thus if you are in workspace \tmp\xyz and issue

]load **ω**\ABC

it will load file \tmp\ABC.dyalog into your current namespace.

The command **]explore** has been removed and replaced by **]open**. See below for details.

Errors are now generated in the 900 range unless they are APL errors (like 22=FILE NAME error). For example what might have previously appeared as error 11 will now show up as 911.

Parsing rules

Command arguments can now include the attribute **S** for *short*, meaning that shorter arguments can be used. For example if *Parse* as been set to **3S** it means that 3 or fewer arguments can be entered. Note that this does not interfere with the *long* attribute and a command could have arguments both *short* (max number of arguments) and *long* (arguments past the maximum are merged into the last one).

User Commands changes

]COMPARE

Now compares a version with the previous one when only a single version is supplied with a single file. Previously any version number given was compared with the LAST one. When you did

```
]compare  myfile  –version=3
```

It would compare V3 with the LAST version, it will now compare V2 with V3.

It also now accepts 0 or negative version numbers to signify "from the last version". For example, assuming <myfile> has 9 versions, doing

```
]compare  myfile  –version=¯1
```

Will compare V7 with V8 (¯1). Note that V0 is considered to be the last one.

]FFIND and]FREPLACE

These commands now search for a regular string and the new switch *–regex* must be used to search using a regular expression.

]FNCALLS

This command now accepts switch *–isolate* to return the name of the objects used by a given function (the argument). Moreover, if *-isolate=**newname*** is used the objects are copied into namespace **newname**. This allows you to modularize code by isolating parts into namespaces.

Note that this is (still) not foolproof and code that uses non standard techniques is likely to be missing items and manual intervention will be necessary.

]LIST

It now displays the number of versions when *–version* is NOT used. The *–version* switch still displays all versions available. Note that the raw form will have a negative number to differentiate it from a real version number (e.g. if 5 versions exist and *–raw* was specified then the "versions" column will contain -5 for that file)

]REMOVEVERSIONS

Now accepts switch *–noprompt* to avoid being prompted for confirmation.

]SETTINGS

Settings now has a new element named **newcmd** which can take 1 of 2 values: 'auto' or 'manual'.

When manual is selected new commands are not detected automatically and you must issue the command]URESET to make new commands effective. This allows systems with slow networks to centralize all user commands without being penalized when someone mistypes a command (in which case the framework will search the (seemingly new) command on the slow system).

The default, 'auto', is the same as it was before: search automatically when a new command name is entered.

]SNAP

Snap accepts a new switch *–nstofolder* to signify "save functions and variables in namespaces in a separate folder". Each namespace's contents is then saved in a different folder with a name similar to the namespace' name. For example if your workspace contains 1 function <Foo> and 1 namespace 'NS1' which contains 1 function <Goo> and 1 variable 'ABC', doing

```
]snap  \tmp  -nstofolder
```

Will produce 3 files:

\tmp\foo.dyalog
\tmp\ns1\goo.dyalog
\tmp\ns1\abc.dyalog

]SUPDATE

Now accepts switch *–DONTSAVE* to prevent automatic saving of the script if it is SALTed.

]UCLEAN

now recurses from the current namespace and function □SE.SALTUtils.cleanWS on which it is based now takes an argument: where to start from (usually #)

]UNEW

Now creates a :Namespace instead of a :Class. The form has also changed a little and fields reordered.

]UVERSION

This command now accepts the name of a file representing a workspace in which case it returns the minimum Dyalog version necessary to)LOAD it.

]WSLOC

Is now called **WSLOCATE**. It also no longer accepts *dll* as argument to –pattern, i.e. *–pattern=dll* will now be refused.

New commands

]ASSEMBLIES

Show all assemblies currently loaded into memory.

]CFCOMPARE file1 file2

This command compares two component files.

]DMX

This command will provide information on the latest APL error

]EFA (Windows only)

This command will associate file extensions .DWS, .DYAPP and .DYALOG with a specific interpreter. This is useful when several versions of APL are installed on a given machine and you wish to change the actual setting.

]OPEN file

Open a file using the default program, or a specified program. This command replaces **]explore** which used to only open SALT files.]OPEN uses .Net to find the program to use to open the file. This can be superseded by using –with= to specify the program to use to open the file. Note that if you specify a relative path the behaviour will be as if using **]explore** and the file will be considered to be a SALT file.

]UUPDATE

This command will bring your current version of tools (e.g. SALT/UCMDs) up to date. This is experimental and is likely to change.

User Commands internal changes

When a command is called it resides in a namespace inside the UCMD framework.

The UCMD framework now sets a few variables in the calling space of your command.

Those are:

RIU a Boolean value indicating whether the **R**esult **I**s **U**sed (e.g. assigned to a variable)

SourceFile the name of the file where your command came from

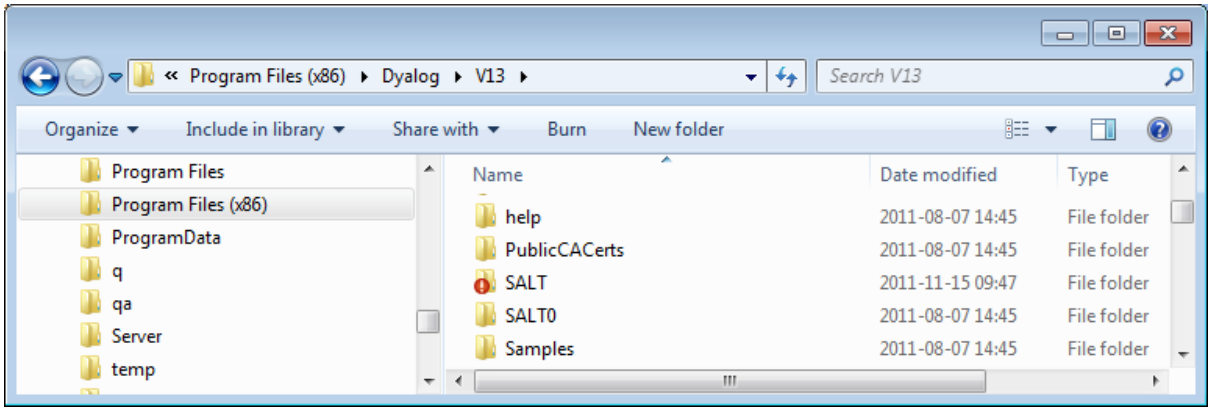
THIS a reference to the calling namespace

These are accessible by using ##, e.g. ##.THIS

Upgrade procedure

Upgrading from a prior version is **not** recommended.

Nevertheless, if you wish to upgrade from version 13 of SALT it is best to remove or rename the previous version and install the new one in its place, e.g.:



Here the previous version has been renamed 'SALT0' and the new version named 'SALT'.

Assuming SALT is enabled you should be able to do

```
SE.SALT.Reboot
```

This will bring in the new SALT but will leave some references pending and you should issue a second

```
SE.SALT.Reboot
```

to resolve this. Then save your session file for subsequent uses of Dyalog.

There is no guarantee this will always work and it is best to upgrade your entire system to V13.1 to ensure proper use.

Dyalog 2012