DYALOG

The tool of thought for expert programming

Dyalog SALT Release Notes

SALT Version 2.40

Dyalog Limited

Minchens Court, Minchens Lane
Bramley, Hampshire
RG26 5BH
United Kingdom

tel: +44(0)1256 830030
fax: +44 (0)1256 830031
email: support@dyalog.com
http://www.dyalog.com

Version 2.40

Revision: 20140609_240

Contents

1 About This Document

This document describes the changes and new features in SALT version 2.40 (released with the first release of Dyalog version 14.0) compared with SALT version 2.31 (released with the first release of Dyalog version 13.2).

1.1 Compatibility with Dyalog Versions

SALT version 2.40 is compatible with all supported versions of Dyalog (that is, 13.1, 13.2 and 14.0), but requires the user command framework to be version 2.0 (this is the version that is shipped with Dyalog version 14.0).

If you are using Dyalog version 13.1 or 13.2, then the user command `]UUpdate` can be used to upgrade SALT and user commands. The default behaviour of `]UUpdate` is to update within the user command framework major release. However, as the user command framework version 2.0 is a major change involving the renaming of many user commands and other behavioural changes, you must specify the `-version=2` modifier to explicitly request an upgrade to the user command framework version 2.0; not including this modifier will limit the upgrade to the latest update prior to this major version change, that is user command framework version 1.34 and SALT version 2.39).

You need to be running with administrator rights for the above command to succeed.

2 Functional Changes

This chapter details the changes made to SALT for version 2.40.

2.1 Tracking Events

A new session parameter, *track*, can now be specified for the `Settings` SALT function. This enables events to be tracked. For example:

```
]SE.SALT.Settings 'track'
```

By default this session parameter is empty. Setting it to have a value of `atinfo` retrieves the function, user and timestamp information (as recorded by the monadic system function `⎕AT`) pertaining to the last time that the function was saved. The information is reinstated when a function is loaded into the workspace by SALT. In previous versions, functions would always report the load timestamp as the timestamp when the function was last loaded into the workspace.

The `atinfo` option can only be used for traditional functions and operators.

> The `atinfo` option is only valid when using SALT version 2.40 in Dyalog version 14.0.

2.1 Classic to Unicode Conversion

Three Unicode glyphs that are used by Dyalog are not available in classic mode; specifically, the glyphs for the *key* function (⌸), *variant* (⌺) and *rank* operator (⍤). In classic mode, these primitives are represented using a `⎕Uxxxx` convention (where `xxxx` is a 4 digit hexadecimal number). For example, the Unicode expression `A(+⍤0 1)B` is displayed in classic mode as `A(+⎕U2364 0 1)B`. However, this syntax is not allowed when using a Unicode interpreter.

To allow smooth interoperation between the classic and Unicode modes, the SALT functions `Save`, `Snap` and `Load` now automatically perform the translation between the `⎕Uxxxx` form in a classic workspace and the Unicode glyphs in source files when saving/loading scripts. To disable this automatic translation (either for performance reasons or to maintain an exact match between on-screen code and code in file), a new session parameter, *mapprimitives*, can now be specified for the `Settings` SALT function. For example:

```
]SE.SALT.Settings 'mapprimitives 0'
```

By default, this session parameter is set to 1. A value of 0 (zero) disables translation and causes the APL interpreter to fail with a TRANSLATION ERROR if these Unicode glyphs are present in a script that is loaded into a classic interpreter or generates SYNTAX ERROR if the `⎕Uxxxx` form is executed in a Unicode environment.

This impacts the `Load`, `Save` and `Snap` functions.

2.2 Namespace Scripting

The meaning of the `-convert` modifier, which controls the conversion of unscripted namespaces into scripts when using the `Snap` function, has changed.

Prior to this release, the `Snap` function always saved an unscripted namespace as a single source file. The `-convert` modifier determined whether the namespace was also converted into script form within the workspace so that subsequent changes to the namespace would be tracked by SALT.

The `-convert` modifier now controls whether SALT creates a single source file for the namespace or saves the namespace as a folder containing one file for each APL object within the namespace (recursively if the namespace contains namespaces).

In addition, all changes to APL objects are tracked by SALT after the `Snap` function has been used irrespective of whether the `-convert` was specified.

These changes mean that the `-nstofolder` modifier is no longer required by the `Snap` function, and it has been removed.

2.3 Execution of .dyalog and .dyapp Files

The `Boot` SALT function can now execute a **.dyalog** script file comprising a single niladic or monadic traditional function without the use of the `-function` modifier.

Files with the **.dyapp** extension can also now contain a niladic or monadic function; double-clicking on these files allows *bootstrap loading* of a Dyalog application.

2.4 Relative Paths

Most SALT functions require the file on which they are to act to be specified by providing a path and filename. The path can either be an absolute path or a relative path following a specific convention.

A new relative path syntax of `..\`<relative path starting from the directory that is the parent of the current directory> is now supported. To identify the directory that is the parent of the current directory, enter the `]CD` user command – the value returned is the absolute path to the current directory. This, when truncated by one level, can be replaced in your absolute path by `..`.

For example, if `]CD` returns a value of `c:\Users\Andy`, then `..` is `c:\Users`.

The `⍵\`<relative path starting from the directory containing the active workspace> syntax has been deprecated and replaced by `[ws]\`<relative path starting from the directory containing the active workspace>. Although the deprecated syntax is still supported in this version of SALT, support will be removed in a later version and Dyalog does not encourage its use.

2.5 SALT Workspace

Prior to this release, the SALT workspace **SALT.dws** could be used to enable and disable user commands. This ability has been removed, and the workspace should now only be used for enabling/disabling SALT.

2.6 Importing Namespace System Variables

When using the `Load` function with the `-disperse` modifier but no modifier value, the values of `⎕CT`, `⎕FR`, `⎕IO`, `⎕ML`, `⎕PP` and `⎕WX` are now imported into the target namespace along with all objects in the specified namespace.

2.7 Specifying Multiple Directories

Two session parameters defined with the `Settings` function specify the full path to the directory or list of directories from which to retrieve user commands (the *cmddir* session parameter) and files (the *workdir* session parameter). The separator used when defining a list of directories is now a `∘` character (the `;` character is still supported for legacy reasons).

2.8 Summary of Modifier Changes

2.8.1 Renamed Modifiers

Some of SALT's modifiers have been renamed to better describe their purpose. These modifiers are detailed in Table *1*.

Table 1. Changes to modifier names

SALT Function	Old Modifier Name	New Modifier Name
`Compare`	`-use`	`-using`
`Compare`	`-zone`	`-window`
`Open`	`-with`	`-using`
`Snap`	`-pattern`	`-patterns`

2.8.1 Deleted Modifiers

Superfluous modifiers have been deleted from SALT functions. These modifiers are detailed in Table *2*.

Table 2. Removed modifiers

SALT Function	Deleted Modifier
`Boot`	`-function`
`Snap`	`-nstofolder`
`Snap`	`-vars`