APL64 Project Q&A Document

Summary of the APL64 Project Q&A Document

This document addresses customer questions and suggestions about the APL64 Project. The APL64 Project team's comments are based upon current available knowledge and are subject to future modification. This is a dynamic document that will be updated with additional information as customers submit questions and suggestions or when additional APL64 Project features are implemented. Review this document frequently to learn more about the APL64 Project.

The APL64 Project Technical Document accompanied the announcement of the APL64 Project. It is available for download here. Update your copy of this document frequently as it is subject to change. In responding to customer comments and questions, the APL64 Project team will sometimes refer to topics in this document.

The APL64 Project team thanks our customers for the questions and suggestions submitted thus far. We encourage customers to provide your comments and will monitor and respond to posts on the APL2000 APL64 Forum and to emails sent to apl64project@apl2000.com.

Note 1: [Updated] will appear for any entry in the Table of Contents that contains new information since the topic was originally presented. The new information also appears in the color red.

Note 2: All new topic entries appear after the Table of Contents entry "Topics Release Date: Oct 29, 2018".

Contents

Topics Release Date: Oct 15, 2018

What are the APL64 Project Priorities?

As of October 2018, the APL64 Project is not a product being licensed by APL2000. Currently, it is a proof of concept that a new version of APL incorporating up-to-date features and scope can be implemented which is also significantly compatible with APL+Win. Based upon customer support and when the APL64 Project team deems it appropriate, alpha/beta versions of the APL64 Project will be published for customers to evaluate.

Initially the priority of the team will be to complete the 'APL+Win compatibility' phase of the project. A design criterion of the APL64 Project is to provide transparent compatibility for applications developed in APL+Win to be run in the Windows environment using the APL64 Project.

Where possible, within the design criteria of the APL64 Project, substantially all APL+Win features have been implemented or suitable alternatives provided in the APL64 Project.

An additional design criterion of the APL64 Project is that the interpreter will be compatible with various operating systems including Windows, iOS, Linux and Android. Certain APL+Win features, e.g. the ActiveX and the □WI Win32 interfaces are not supported by the iOS, Linux or Android operating systems. These interface components are included with the APL64 Project, but are not required unless a programmer utilizes them in an APL64 Project application. In that case, the application will be compatible with the Windows operating system, but will not be compatible with the iOS, Linux or Android operating system.

After the fullest possible, transparent compatibility with APL+Win is achieved in the APL64 Project, additional priorities will be considered. Customer support is paramount in driving development. As much as feasible, the APL64 Project direction will follow customer needs and requests.

Will the APL64 Project Support □WCALL (Win32 APIs)?

Yes. □WCALL is fully supported in the APL64 Project.

Will the APL64 Project Support □CALL (ASMFNS)?

No. The APL64 Project 'cross-platform' design criterion requires that the interpreter portion of the project must operate in the Windows, iOS and Android operating system environments. To satisfy this design criterion and have a single, cross-platform interpreter code base, the APL64 Project interpreter must be implemented using the Microsoft .Net Core framework.

The .Net framework executes application code in a virtual machine which protects the operating system and the physical memory of the workstation from malware. Thus, the .Net framework does not permit the execution of assembly language programs as this act upon the physical memory of the workstation and are deemed 'unmanaged' or unsafe code. Therefore, it is not possible to directly support the APL+Win □CALL system function in the APL64 Project.

Generally, the □CALL functionality of APL+Win was utilized to improve performance. The APL64 Project interpreter has many more opportunities for performance improvement which have been identified by the APL64 Project team and probably even more which will be identified in the future.

Several alternatives to □CALL are available now:

(a) The APL64 Project includes an efficient □NA Extension Function mechanism that is described in the "Access C# Methods in .Net Assemblies" topic of the "APL64 Project Technical Document" here. Using this feature customers may develop their own C# extensions to the APL64 Project.

(b) An on-going effort is underway in the APL64 Project to port all □CALL-based functions in the APL+Win ASMFNS.W3 workspace to corresponding □NA extension functions in a companion .Net assembly containing the implementation code.

Will the APL64 Project Support the APL Grid ActiveX Component?

Yes. The APL Grid is a Win32 ActiveX component that is fully supported in the APL64 Project and the syntax is the same as in APL+Win, which is ideal for APL64 Project programmers targeting the Windows operating system environment.

APL64 Project programmers targeting the iOS, Linux, Android or other non-Windows environments should note these environments do not support ActiveX. Therefore, if the APL64 Project programmer includes the APL Grid in their application, it would not be suitable for these non-Windows operating system environments. There are commercially-available Excel-like grid components that are available for the .Net Core environment, e.g. SpreadsheetGear that could be used in an APL64 Project application targeting iOS , Linux, Android or other non-Windows operating system environments.

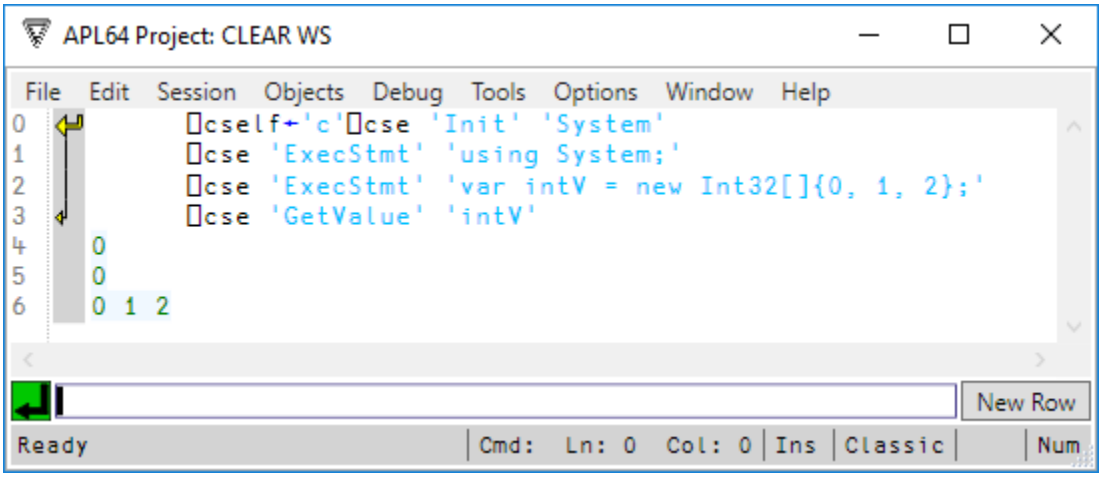
Will the APL64 Project Support the APL2000 Draw ActiveX Component?

Yes. The APL2000 Draw ActiveX component is a Win32 ActiveX component that is fully supported in the APL64 Project and the syntax is the same as in APL+Win, which is ideal for APL64 Project programmers targeting the Windows operating system environment.

APL64 Project programmers targeting the iOS, Linux, Android or other non-Windows environments should note these environments do not support ActiveX. Therefore, if the APL64 Project programmer includes the APL2000 ActiveX component in their application, it would not be suitable for these non-Windows operating system environments.

Will the APL64 Project Support the APLNext C# Script Engine (□CSE)?

Yes. The APL64 Project supports □CSE with the same ActiveX interface and syntax as in APL+Win. This is demonstrated in the graphic below:



It is plausible that the APL64 Project could support a more direct interface for □CSE since the APL64 Project was developed using .Net, and thereby possibly improving the performance of the 'direct' interface between it and the APLNext C# Script Engine. However, the priority for this implementation would depend on the level of expressed customer support for this enhancement.

Will the Installation of the APL64 Project Require Activation?

Activation or similar measures are necessary for the developer portion of the APL64 Project so that only the customers who support the product and the intellectual property on which it is based will benefit from it.

Here are some options which are being considered for the APL64 Project:

1. Activation via the Internet at installation time and license renewal
2. Telephone activation when an Internet connection is not available
3. Internet log-on analogous to that in Microsoft Visual Studio whenever the developer version is used

Will the Installation of the APL64 Project be 'Easy'?

Defining 'easy' is never easy. For APL programmers a 'development' version of the APL64 Project will be installed with a Windows 'installer' program. For deployed application systems created using the 'development' version, a 'runtime' version will also be available.

As indicated in the APL64 Project Technical document, the installation folder of the APL64 project in the Windows operating system environment may be any workstation folder with the appropriate user permissions. Unlike APL+Win, the APL64 Project may be installed to the '...\program files' folder.

Will the APL64 Project Cause "Disruption" to Current Applications?

No. Customers using APL+Win may continue to do so if desired. APL2000 support for the APL+Win product will continue as long as there is significant customer support for it.

An important design criterion of the APL64 Project, which has been achieved, is for significant compatibility with APL+Win applications running in the Windows environment. For many APL+Win applications, the APL+Win workspace may be directly loaded into the APL64 Project using the)LOAD system command.

Will the APL64 Project Extend the Range of APL+Win Primitive Operators?

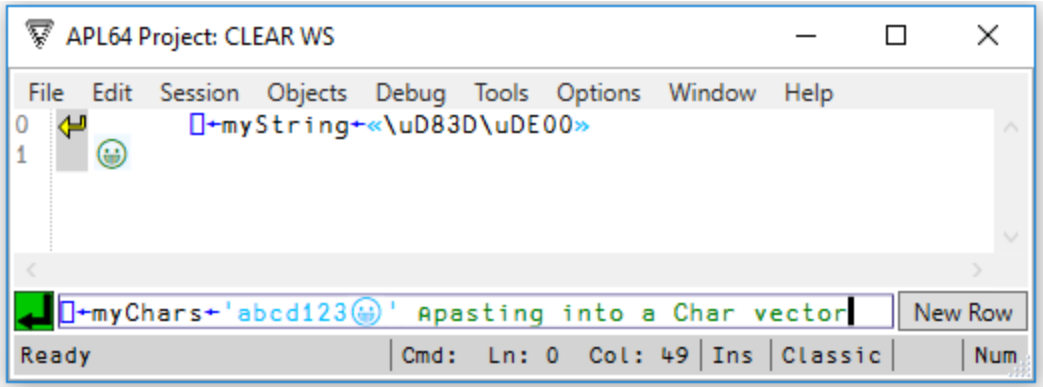
Yes. A design criterion achieved for the APL64 Project is that it will be possible to more easily create additional primitive operators than in APL+Win. So additional primitive operations may be implemented depending on expressed customer interest.

Will the APL64 Project Support Unicode Characters?

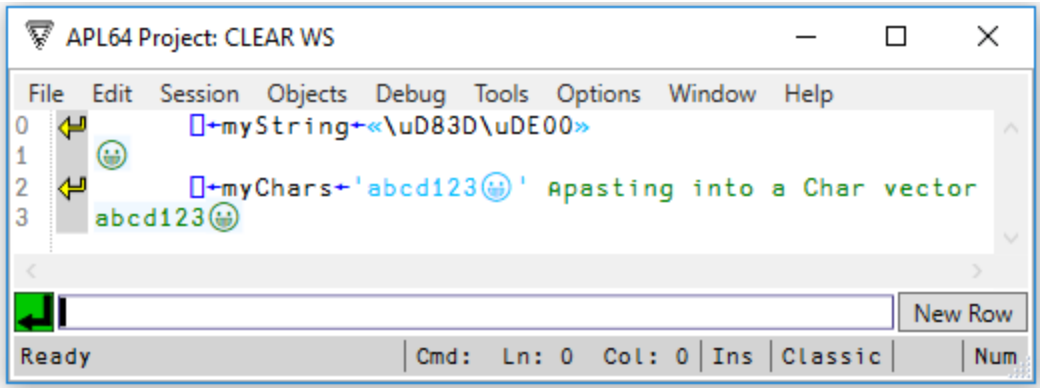
Yes. Unicode characters are fully supported by the APL64 Project. They can by typed or pasted directly into any character or string values in function definitions or at the command line. Unicode characters can also be used in comments. For string values, the escape sequence \uXXXX can be used to enter any Unicode character via its 4-digit hexadecimal code. Unicode characters with hex values outside the Basic Multilingual Plane (0x0000-0xFFFF) can be entered as a surrogate pair. This is handled automatically via pasting or keyboard entry.

For example, the smiling face emoji 😀 is at Unicode character 0x1F600 which may be entered as surrogate pair \uD83D\uDE00 or simply pasted into a string value. For example:

Using 'escaped hexadecimal Unicode' in an APL64 Project string variable and Selecting a Unicode character and pasting it into an APL64 Project character vector:



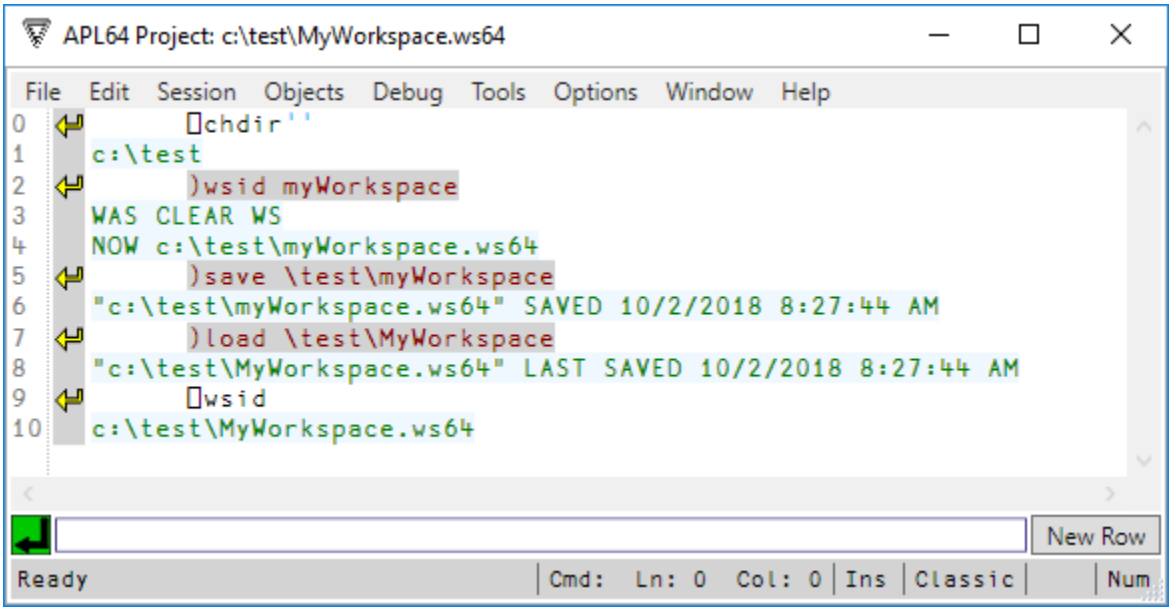
Displaying Unicode characters in the APL64 Project session:



Will the APL64 Project Support Workspace File Extensions?

Yes. For an APL64 Project workspace any workspace file extension, except for '.w3', may be used, but by default the workspace file extension is ".ws64" if not supplied by the user when loading, saving, copying, or storing an APL64 Project workspace.

The APL64 Project always displays the workspace file extension in the output from workspace related system commands and functions. For example:



The ".w3" file extension is reserved for APL+Win. When used as a workspace source name such as in a)LOAD or)COPY command, it designates an APL+Win workspace and that workspace is automatically loaded and converted from an APL+Win .w3 format into the APL64 Project memory. The ".w3" extension is not allowed as a workspace target name in a)SAVE or)STORE command.

Workspace conversions are one-way from APL+Win to the APL64 Project. An APL64 Project workspace contains features and data types that cannot be represented in APL+Win and as such cannot be successfully saved in an APL+Win workspace.

An APL+Win .w3 workspace cannot also be overwritten with an APL64 Project workspace. This prevents accidentally overwriting an APL+Win workspace with an APL64 Project workspace.

Will the APL64 Project be .Net Core Compliant?

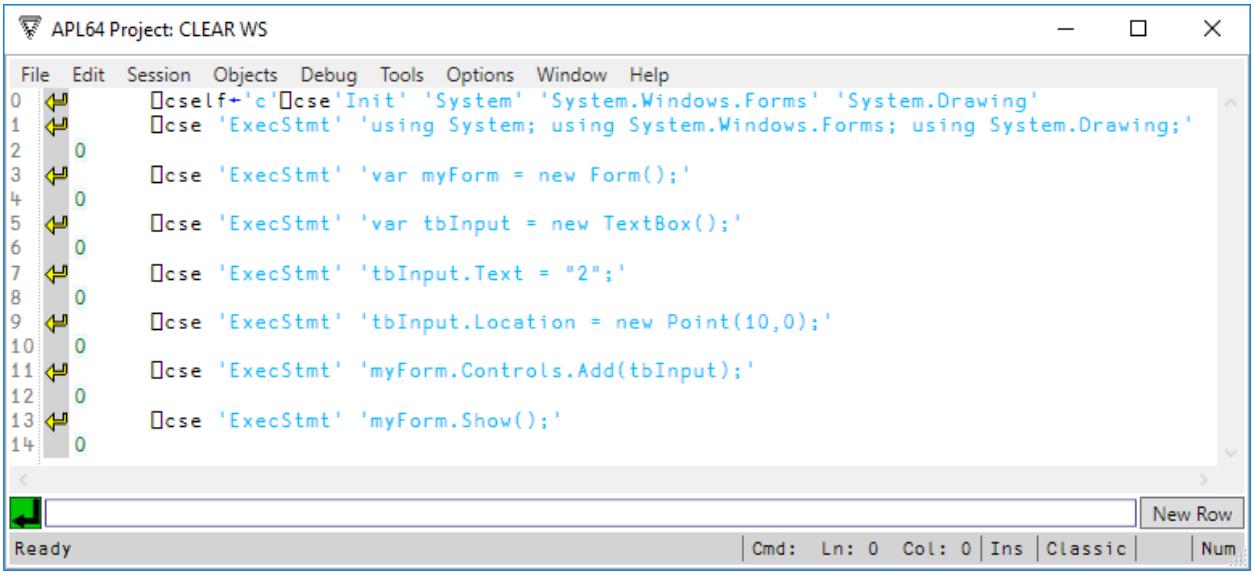
Yes, by design the APL64 Project interpreter is designed to be .Net Core Framework compliant. As of October 2018, compliance with .Net Core means that a subset of the full .Net Framework can be used to implement an application like the APL64 Project. Microsoft has been continuously increasing the content of that subset to dramatically reduce the restriction it may represent.

The reason that the APL64 Project is designed to be .Net Core compliant is that this assures that it will be suitable for the Windows, iOS, Linux and Android operating system environments.

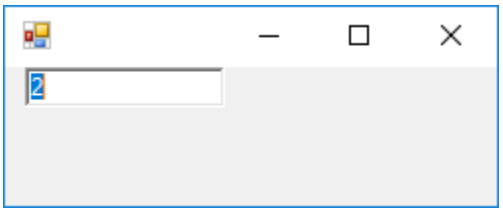
Besides using the APL64 Project .Net Core-compatible interpreter, the programmer developing an APL64 Project application needs to carefully select the technology used. For example, the .Net Core does not support □WI, ActiveX or assembly language use, so these technologies cannot be used in application development targeted to the iOS, Linux or Android operating system environment. The APL64 Project interpreter design permits the programmer to selectively employ technology in a transparent way, i.e. if it is not used in the application, the associated interpreter component does not become part of the application.

Will the APL64 Project Access Microsoft WinForms?

Yes and the syntax is the same as in APL+Win for an APL+Win application to use the .Net System.Windows.Forms namespace (WinForms) GUI development tools with the APL64 Project as the client and WinForms as the server:



Running the above APL64 executable statements generates the following WinForms GUI:



It is also anticipated that in a future version of the APL64 Project the roles of client and server could be reversed, i.e. with WinForms or any .Net project as the client and the APL64 Project interpreter as the server. In this scenario a Microsoft Visual Studio .Net WinForms or WPF project will be able to reference the APL64 Project as a .Net assembly and use it to process information captured and provided to the APL64 Project by the WinForms GUI.

Will the APL64 Project be Compatible with MS Visual Studio?

The APL64 Project has two main components, the interpreter and the APL programmer session (GUI).

A major design criterion of the APL64 Project is to provide a very high level of compatibility with APL+Win and this includes the APL programmer session. The APL64 Project programmer session is designed to provide the classic APL programming experience. The APL64 Project Technical Document provides excellent documentation of the APL64 Project programmer session including the classic format and many significant options and enhancements. Most APL64 Project programmers will use the APL64 Project programmer session to access the APL64 Project interpreter in the classic manner. Used in this manner the APL64 Project does not require the APL programmer to use Microsoft Visual Studio.

Part of the APL64 Project design is that the APL64 Project interpreter may be used independently of the APL programmer session. For example, using the APL64 Project interpreter to perform array processing as part of a larger .Net-based application system. In that scenario, the larger .Net project could be developed using Microsoft Visual Studio and the APL64 Project interpreter would be a .Net assembly which is referenced by that larger .Net project.

The level of expressed customer support for the use of the APL64 Project interpreter independently of the APL programmer session will determine the implementation priority ascribed to this feature.

Will the APL64 Project Provide Support for Object-Oriented Classes?

Inherent in the internal design of the APL64 Project are mechanisms necessary to support user-defined namespaces, classes, objects and 'dot' notation in an object-oriented manner. These object-oriented features will not become available until the 'compatibility with APL+Win' phase of the APL64 Project is complete. The priority for object-oriented features will depend on the level of expressed customer support for these features.

Will the APL64 Project Support Intellisense?

Intellisense is Microsoft's term for auto completion. For example, if the programmer starts typing a name in the Session Command Line, the system might display a list of all function and variable names that contain the sequence of characters typed by the user as part of a name, or when setting the value of an argument, auto completion would provide a list of possible values for that argument from which the user may select.

The core mechanism required to support Intellisense (auto completion) is already built into the Session Command Line and the user-defined function editor of the APL64 Project. Support for Intellisense (auto completion) in the APL64 Project might be explored after the 'compatibility with APL+Win' phase of the APL64 Project is completed.

Will there be Support for Keywords as Substitutes for APL Symbols?

This feature is not available in the current implementation of the APL64 project. The priority for this implementation will depend on the level of expressed customer support for this feature.

Will the APL64 Project Support an Interface to MS SQL?

Yes. For the APL64 Project programmer targeting the Windows operating system environment, the □WI ActiveX interface is available in the APL64 Project and can be used with the Microsoft ADO interface to Microsoft SQL Server databases. Since the □WI ActiveX interface is based on Win32, variable sizes are limited when it is used.

For the .Net programmer, the □NA or □CSE interface in the APL64 Project can be used with the Microsoft ADO.Net interface to Microsoft SQL Server databases.

Will there be Support for NoSQL-Format Databases in the APL64 Project?

Many NoSQL-format databases have application programming interfaces (APIs) which support an ActiveX or .Net interface.

For APL64 Project programmers targeting the Windows operating system environment both the ActiveX and .Net interface pathways are available in the APL64 Project.

To use a NoSQL-format database which provides an API with an ActiveX interface the ⎕WI interface is fully supported in the APL64 Project.

To use a NoSQL-format database which provides an APL with a .Net interface, the ⎕NA and ⎕CSE interfaces are available in the APL64 Project.

Will Timestamps be Added to Variables in the APL64 Project to Show When Last Modified? [Updated]

There are no plans for doing so at this time because it would add execution overhead each time a variable's value is assigned or modified. If there is sufficient interest in having this feature as a debugging aid, it could be implemented and be available when running in debug mode. Customer interest in this feature will be considered in its implementation priority.

Note: Could this expense be ameliorated by restricting variable timestamps to only global variables and not anything that's localized? When the interpreter is running, there is no means to distinguish between a global or local variable. Even if there was such a discriminator, it would add additional processing delays over and above the processing time necessary to update the timestamp information. If there is significant customer interest in this feature, it could be included in the priorities for implementation in the APL64 Project.

Will the APL64 Project be Able to Export .Net Assemblies?

The design for the APL64 Project includes this capability. However, as of October 2018 the highest priority for the APL64 Project is to complete the 'compatibility with APL+Win' phase of the project.

The level of expressed customer interest will determine the priority of implementing the export of .Net assemblies from the APL64 Project.

Will the APL64 Project Support R?

The APL64 Project provides two interfaces suitable for accessing R technology. For APL programmers targeting the Windows operating system environment, the ⎕WI ActiveX interface is available. The Lescasse zREngine can be used with the ActiveX interface and is available here http://www.lescasse.com/Content/zREngine.aspx.

For .Net programmers, the APL64 Project ⎕NA or ⎕CSE interface to directly use .Net source code is available. The RdotNet toolkit can be used with the ⎕NA interface and is available here http://jmp75.github.io/rdotnet/.

Using the APLNext C# Script Engine (⎕CSE) interface, the APLNext R toolkit can be used and is available here.

Will the APL64 Project Support Function Assignment?

This feature is not currently available but might be possible to implement in the APL64 Project in the future. Whether or when this feature is implemented will depend upon expressed customer interest and the priorities associated with the other customer suggestions.

Will the APL64 Project Support Hadoop and Other Cloud Based Environments?

Yes, via the APLNext C# Script Engine (⎕CSE) interface. The APL64 Project action requests [queries with APL64 Project variable values] can be passed to a cloud-based server for processing and the results received by an APL64 Project application system. To learn more, obtain the APLNext C# Script Engine manual here.

Will the APL64 Project Support Versioning Software on Workspaces? [Updated]

Code versioning software, e.g. Perforce, GitHub, includes code manipulation features, e.g. compare, merge, in addition to journalized source code storage.

To take advantage of this type of software in a convenient manner, APL source code format must be exported/imported in a format which is not a proprietary serialization. For example, the APL+Win workspace file format is a proprietary serialization of the workspace content, which cannot be readily handled by code versioning software.

On the other hand, a proprietary serialization of the contents of an APL workspace is compact and secure. For this reason, the APL64 Project can save the contents of a workspace using a proprietary serialization, just like all other APL implementations.

The APL64 Project includes the new)LOAD and ⎕LOAD methods to import an APL+Win workspace into the APL64 Project session memory, run it and save it to the new APL64 Project workspace format.

It is anticipated that the APL64 Project will provide an option to export the contents of a workspace in a format suitable for use in code versioning software. The APL64 Project team has already identified a suitable format based upon their continuous use of the GitHub code versioning software used for the APL64 Project source code. The priority for this implementation will depend on the level of expressed customer support for this feature.

The APL64 Project will support exporting one or more functions in a workspace to a format suitable for code repository software such as GitHub, Perforce, etc. It is not intended that the APL64 Project will integrate with any specific code repository software because that would limit a customer's choices and suggest that APL2000 endorses a particular code repository software vendor. Within this design criterion, no particular code repository software would be prevented from using the output of the APL64 Project.

Will the APL64 Project Support the Power Operator?

This would not be a priority in the initial release of the APL64 Project. The priority for this implementation in the future will depend on the level of expressed customer support for this feature.

Topics Release Date: Oct 29, 2018

Can the APL64 Project Interpreter be used as a Component of an Application System?

APL+Win provided virtually all the application system components beyond the operating system for most existing APL+Win applications. Therefore, APL-based application systems developed with the APL64 Project will do the same.

In addition, the APL64 Project interpreter is based upon the .Net Core Framework, so it can also be used as a sub-component of a larger application system and deployed to the Windows, iOS, Linux or Android operating system environments. The APL64 Project interpreter is design to interoperate with other .Net programming languages like C# or VB.Net targeting a cross-platform environment.

What Will be the Product Name of the APL64 Project?

The name of a potential APL64 Project production version has not yet been identified. Customer input will be considered should an APL64 Project product be developed.

Will □SYSID and □SYSVER be Supported in the APL64 Project?

Yes. However, the current values are not yet publicly available.

Are the Panes in the APL64 Project Session Graphical User Interface (GUI) Floatable and Resizable?

Yes. Unlike APL+Win, which constrained all GUI components within the main APL+Win window, the individual panes of the APL64 Project session may be docked, floated, resized and independently positioned on any display monitor connected to the workstation.

Will the APL64 Project Support Full Functional-Programming?

It may be possible to incorporate functional programming technology into the APL64 Project, provided it can be done in a manner that does not diminish the traditional APL syntax and function structures. Just as in APL+Win, code snippets can be pasted into a user-defined APL function.

What Will Be the Pricing for an APL64 Product?

No licensing or pricing decisions have been finalized at this time. APL2000 is currently analyzing the APL64 Project Survey responses and continues to welcome input on the APL64 Project. This will inform APL2000's decision on how to proceed. Decisions will depend upon the level of customer support for the APL64 Project.

Will the APL64 Project Consider Removing Functions Since a Streamlined Interpreter Will Run Faster?

By design the APL64 Project interpreter is not monolithic. The programmer can select to include various components, including the native and APL component file systems and the □WI interface, in an application system. Only when they are included by the programmer, will those components be loaded into memory.

Will the APL64 Project Include Unlimited Runtime?

Yes, the APL64 Project would preserve the 'unlimited runtime' licensing model for the distributable components of the APL64 Project.

Will there be a Multi-Threaded Version of the Each Operator?

This kind of feature is also sometimes called Parallel-Each and it is a definite possibility but it is not something that would be a high priority in an initial release of the APL64 Project. The priority for this implementation will depend on the level of expressed customer support for this feature.

There are a number of important design questions that must be resolved before anything like this can be brought into production. A key issue with trying to do this via an extension of the Each operator is that this doesn't provide a mechanism for specifying execution attributes to control how a particular invocation should be handled. For example, should the parallel execution take place only on the local machine or should it be distributed across multiple machines? How many threads can be allocated to the parallel execution? Can these parallel execution threads themselves contain nested invocations of parallel-each? If multiple machines are to be used, at what element count threshold should multi-machine execution be triggered? Are the parallel threads allowed to have side-effects such as modifying the state of global variables? If multiple APL64 Project instances are concurrently running parallel-each operations, how should these competing workloads be prioritized relative to each other? How would exceptions and interrupts be handled?

Because of these many open design questions, and the lack of ability for a simple operator to express per-invocation control over such control parameters, it might be fruitful to explore alternative, more expressive means of handling parallel execution such as extension of Web Services to the APL64 Project or via extensions to the :FOR loop (perhaps via a :FORK loop with optional control parameters, for example.)

Will the APL64 Project Provide a 'Web Services' Component?

Expressed customer interest will determine if there will be a 'web services' component for the APL64 Project. The design of the APL64 Project is such that the APL programmer can decide to use certain components or not. This applies to a 'web services' component. Some APL programmers will utilize it and others will not do so. Therefore, making these components irrevocably-integrated elements of all APL64 Project applications would not be in the best interest of APL2000 customers.

Implementing this feature for the APL64 Project would provide a way for the programmer to expose application-specific functions as web services which could be incorporated into the programmer's choice of web server. This provides the greatest flexibility to the programmer, because web server choices vary according to the operating platform.

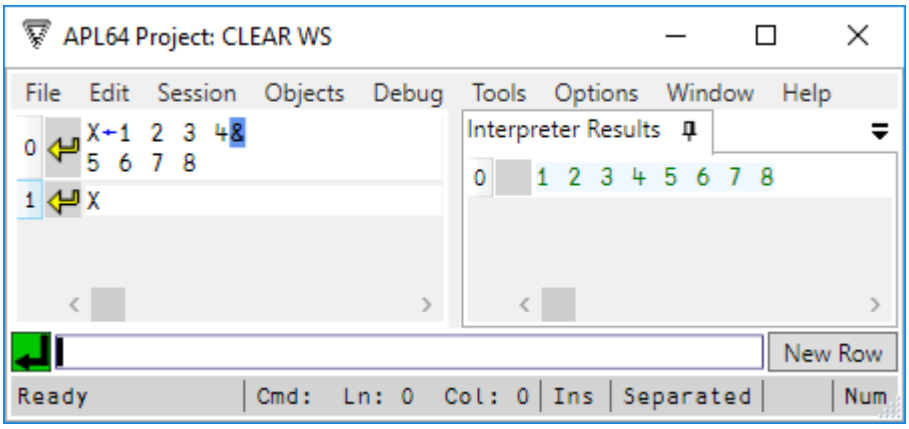
Is a Version of the APL64 Project Available to Try?

No licensing or pricing decisions have been finalized at this time. APL2000 is currently analyzing the APL64 Project Survey responses and continues to welcome input on the APL64 Project. This will inform APL2000's decision on how to proceed. Decisions will depend upon the level of customer support for the APL64 Project.

Could a Different Glyph be Selected for Continuation Lines Rather than Ampersand (&)?

Yes. However suggestions from customers will be necessary to consider this idea.

In the APL64 Project an APL executable statement in the Session Command Line or a line in a user-defined function editor instance can be continued by entering an ampersand as the last, non-blank character on the row and continuing the remainder of that statement or line on subsequent rows.



The current implementation of the ampersand as a continuation glyph in the APL64 Project is a niladic use which is compatible with potential other contexts. For example it would be possible to implement a new APL64 Project syntax to be applied when the ampersand prefixes a non-blank object name.

How are APL Variables (Including Nested) Edited in the APL64 Project?

The APL64 Project provides a new unified browser/editor for APL64 Project variables which may be simple, i.e. homogeneous or rank 2 or less, or complex, i.e. possibly non-homogeneous and nested. This browser/editor component of the APL64 Project is an area of intensive design and testing, so the details are subject to change prior to its release.

When the unified browser/editor is applied to an APL nested array, the user can navigate through the nesting levels until a 'simple' element is identified. The selected simple element, e.g. a homogeneous array of rank 2 or less can then be viewed or edited directly within the unified browser/editor.

The unified browser/editor also includes options to apply transformations to a nesting level of the array such as undo/redo, reshape, rotate, delete, insert, paste, copy and change data type.

Customer input from users of the unified browser/editor will certainly provide significant suggestions for enhancements.

Will the APL64 Project Include Anonymous and Lambda Function Structures as an Option to the Classic, User-Defined APL Function?

The APL64 Project tokenizer and parser have been modified to recognize these constructs, but completing the implementation will not occur before the 'compatibility with APL+Win' phase of the APL64 Project is complete. Also, when or if features such as these were to be included in the APL64 Project would depend upon the level of expressed customer interest.

Will the APL64 Project Include Support for User-Defined Operators?

The current design of the APL64 Project does not preclude the implementation of user-defined operators, but it is not a feature which is anticipated to be in an initial release of the APL64 Project. Expressed customer interest in this feature will also affect the resources applied to this enhancement.

There are existing models, e.g. IBM APL2 and Dyalog, for this feature, so the APL64 Project may be able to identify the best ideas from them. Because of APL64 Project-specific function argument enhancements under development, the syntax for creating a user-defined operator may not follow these models precisely.