Welcome to APL 2010 LPA Berlin

Content of the stick	
Goto	**Further Hints**
Table of contents and links to contributions	Links to complete contribution are available in the table of contents and from the heading of the summary To view some contributions correctly the fonts in the folder Fonts must be installed on your computer!
Further content	Additional Information on a paper and supplied software
Conference locations	
Sponsor information	Advertisments and links to home page
Schedule	Weekdays link to schedule of the day
Daily schedule	

Conference Program APL 2010 LPA - Berlin: Table of Contents

Topic 4: Challenging (production type) applications solved with array processing languages

Topic 5: Theory and practice in array processing language design and implementation

Reviewed Short Communications

Topic 4: Challenging (production type) applications solved with array processing languages

Topic 5: Theory and practice in array processing language design and implementation

Tutorials

Workshops

Forum

Selling Tutorials

Vendor Sessions

APL2000

V02: Patrick Parks (APL2000): APL+Win V10 Enhancements in Detail
V03: Patrick Parks (APL2000): APL+Win V10 Interpreter Performance Enhancement in Detail
V04: Joe Blaze (APL2000): APLNext Supervisor — A Simple Example
V05: Joe Blaze (APL2000): APL2000 Customer Forum
V06: Joe Blaze (APL2000): APLNext VisualAPL — Programming Examples
V07: Joe Blaze (APL2000): APLNext WebServices — A Practical Example
V08: Joe Blaze (APL2000): "MVC" and "Presentation Model" System Architecture for APL

Dittrich & Partner Consulting

V09: Klaus-Peter Friedrich (Rheinischer Sparkassen- und Giroverband (RSGV)): STS.win - An APL2 OLAP Database (Goto paper)
V10: Katrin Holzmüller and Vladimir Zakgeym (DPC GmbH): APL2 at a Young Glance

Dyalog Ltd.

V11: John Daintree (Dyalog Ltd.): Taking APL for a RIDE
V12: Morten Kromberg (Dyalog Ltd.): Dyalog Technical Keynote
V13: Jay Foad (Dyalog Ltd.): An interpreter for Vanilla Siteswap
V14: Morten Kromberg (Dyalog Ltd.): Your Application as an SQL Data Source
V15: Kai Jäger (APLTeam): APL2XML
V16: Stig Nielsen (SimCorp A/S): Migrating SimCorp Dimension to Dyalog APL Unicode
V17: Ryan Tarpine and Mstislav Elagin: Winning the Dyalog Programming Contest 2010

IBM APL Products and Services

V18: David Liebtag (IBM APL Products and Services): Using APL2 with Java and the WebSphere Application Server
V19: David Liebtag (IBM APL Products and Services): Recent APL2 Enhancements

Goto top of page Goto top of table of contents

Further content of the stick

Author	Contribution	Add on
Brian Becker	APL and Web Services	see Folder SAWS
Devon McCormick:	Parallel Programming Theory and Examples Towards Sketching a Taxonomy for Problem Estimation	Examples of Parallel Code in J
Markos Mitsos:	Building reference systems in German health insurance	Presentation
	Kx	see Folder Kx
Dr. Reiner Nussbaum:	Hash arrays as Dyalog APL objects	see Folder HashArrays
Bob Smith:	NARS2000 software	see folder NARS, download current version

Goto top of page

Conference Locations

Lecture Rooms			
Type	**Building**	**Room No.**	**Reference to Schedule**
Plenary Lecture Room	Main building	H1058	All plenary sessions (Welcome, Vendor Forums, IP, KN, FO, Closing) and all sessions in the first column of the schedule
Second Lecture Room	Main building	H0107	All sessions in the second column of the schedule
Third Lecture Room	Main building	H0110	All sessions in the third column of the schedule
Forth Lecture Room	Main building	H0111	All sessions in the forth column of the schedule
Fifth Lecture Room	Main building	H0112	All sessions in the fifth column of the schedule

Main Building	Straße des 17. Juni 135

Conference Facilities				
Type	Building	Location		Reference to Schedule
		Monday to Wednesday	Thursday	
Conference office	Main building	Patio	In front of H0107	Beginning of Conference Check-In
Exhibition area	Main building	Patio		
Refreshments, meals	Main building	Patio	In front of H0107	Coffee Break, Lunch

Goto top of page Open map as pdf Goto Google Maps

Goto top of page

Sponsors of APL 2010 LPA

We would like to thank all our sponsors for making this big event possible and for enriching APL2010 with valuable input, presentations and more. Visit their advertisments (1st column) or home page.		
Allianz	http://www.allianz.de	Sponsor of the sticks
APL2000	http://www.apl2000.com	Gold Sponsor
Dittrich & Partner Consulting	http://www.dpc.de/	Silver Sponsor
Dyalog Ltd.	http://www.dyalog.com/	Gold Sponsor
IBM APL Products and Services	http://www.ibm.com/software/awdtools/apl	Gold Sponsor
Kx	http://kx.com	Silver Sponsor

Goto top of page

Conference Program APL 2010 LPA - Berlin: Schedule

Conference Program APL 2010 LPA - Berlin: Time Schedule																
	Monday, September 13		Tuesday, September 14					Wednesday, September 15					Thursday, September 16				
9:00-10:00	9:00 Beginning of Conference Check-in 9:30 Gerald Dittrich: Welcome		P08	P16		V18	V11	P14	P20	WS3	V03	Dyalog Vendor Session	P06	ST1		V07	V15
10:00-11:00	IP1: Dr. habil. Sven-Bodo Scholz: Multi-/ Many-Cores: array programming at the heart of the hype!		P09	P17		V19	V12	P15	SC2 SC3		V04		P07	ST2	GSE WG APL meeting	V08	V16
11:00-11:15	*Coffee Break*																
11:15-12:15	KN1: Helmut Weber: MultiCore & Hybrid Systems - New Computing Trends ?		IP2: Dr. James A. Brown: APL and its Influence on Modern Computing					KN2: Prof. Dr.-Ing. Horst Zuse: The Origins of the Computer					FO2: Dr. James A. Brown (Chair): The Future of Parallel Computing with APL				
12:15-13:15	*Lunch*																
13:15-14:15	IBM APL2 Vendor Forum		Dyalog Vendor Forum					APL2000 Vendor Forum					DPC Vendor Forum				

14:15-14:30	*Coffee Break*															
14:30-15:30	P04	P12	TU1	WS1	Dyalog Vendor Session	P10	P18	TU3	WS2	V01	P02	TU4	WS4	V09	V05	Closing Session incl. V17
15:30-16:30	P01	P13	TU2			P11	P19			V02	P03			V10	V06	
16:30-16:45	*Coffee Break*															
16:45-17:45	FO1: Bob Smith (Chair): APL in 2020					P05		GSE WG APL meeting		V13	SC1	TU5		DPC Vendor Session	V14	
17:45-19:00																
19:00	Welcome Party (Boat Trip)									Banquet						

Topics:

1. **State of the art** of array processing languages (P01-P03)
2. Array processing languages in **computer sciences** (P04-P07)
3. **Parallel programming** with array processing languages (P08-P11)
4. **Challenging** (production type) **applications** solved with array processing languages (P12-P15, SC1)
5. **Theory and practice** in array processing language design and implementation (P16-P20, SC2-SC3)

Types:

IPn:	Invited Paper
KNn:	Keynote Speaker
Pnn:	Paper
SCn:	Short communication
TUn:	Tutorial
WSn:	Workshop
FOn:	Forum / Panel
STn:	Selling Tutorial
Vnn:	Vendor Session

Vendor sessions:

APL2000:	V01-V08
DPC:	V09-V10
Dyalog Ltd.:	V11-V17
IBM:	V18-V19

Further venues:

APL2000 runs a browser based tutorial and an APL demonstration at its exhibit table.

Contents to untitled vendor sessions might be announced during the conference.

Goto top of page schedule Monday Tuesday Wednesday Thursday

Conference Program APL 2010 LPA - Berlin: Daily Schedule

Monday, September 13						
9:00-10:00	9:00	Beginning of Conference Check-in				
	9:30	Gerald Dittrich: Welcome				
10:00-11:00	IP1: Dr. habil. Sven-Bodo Scholz: Multi-/ Many-Cores: array programming at the heart of the hype!					
11:00-11:15	*Coffee Break*					
11:15-12:15	KN1: Helmut Weber: MultiCore & Hybrid Systems - New Computing Trends ?					
12:15-13:15	*Lunch*					
13:15-14:15	IBM APL2 Vendor Forum					
14:15-14:30	*Coffee Break*					
14:30-15:30	P04: Joe Blaze: APLNext VisualAPL	P12: Helmut Engelke: Improving Violinists' Intonation	TU1: Bernd Geisselhardt: Development Environment on the Workstation and Runtime			

			Environment on the HOST? It works!	WS1: Brian Becker: APL and Web Services	Dyalog Vendor Session
15:30-16:30	P01: Morten Kromberg, Jonathan Manktelow, John Scholes: APL# - An APL for Microsoft.Net, Mono, SilverLight and MoonLight	P13: Markos Mitsos: Building reference systems in German health insurance	TU2: Patrick Parks: APL+Win Performance		
16:30-16:45	*Coffee Break*				
16:45-17:45	FO1: Bob Smith (Chair): APL in 2020				
17:45-19:00					
19:00	Welcome Party (Boat Trip)				

Goto top of page schedule Monday Tuesday Wednesday Thursday

Tuesday, September 14					
9:00-10:00	P08: Devon McCormick: Parallel Programming Theory and Examples Towards Sketching a Taxonomy for Problem Estimation	P16: Patrick Parks: New APL+Win System Features		V18: David Liebtag: Using APL2 with Java and the WebSphere Application Server	V11: John Daintree: Taking APL for a RIDE
10:00-11:00	P09: Joe Blaze: APLNext Supervisor	P17: Dr. Herman Singer: Succinct - A new APL dialect		V19: David Liebtag: Recent APL2 Enhancements	V12: Morten Kromberg: Dyalog Technical Keynote
11:00-11:15	*Coffee Break*				
11:15-12:15	IP2: Dr. James A. Brown: APL and its Influence on Modern Computing				
12:15-13:15	*Lunch*				
13:15-14:15	Dyalog Vendor Forum				
14:15-14:30	*Coffee Break*				
14:30-15:30	P10: Wai-Mee Ching, Da Zheng: Benefits and Limitations of Array-style Programs for Parallel Execution	P18: Richard Smith: Damage Resistant Component Files Using Journaling and Other Techniques	TU3: Dan Baronet: User Commands in Dyalog APL	WS2: John Scholes: Introduction to D-Functions	V01: Joe Blaze: WPF Presentation & APL Business Rules Components in a Windows Application System
15:30-16:30	P11: Morten Kromberg, Michael Hughes: Parallel Computation Using Peach, Prank and Pouter	P19: Geoff Streeter: Supporting APL keyboards on Linux			V02: Patrick Parks: APL+Win V10 Enhancements in Detail
16:30-16:45	*Coffee Break*				
16:45-17:45		P05: Joe Blaze: APL+Win Interfaces		GSE WG APL meeting	V13: Jay Foad: An interpreter for Vanilla Siteswap
17:45-19:00					
19:00					

Goto top of page schedule Monday Tuesday Wednesday Thursday

Wednesday, September 15					
9:00-10:00	P14: Lars Wentzel: CPAM — Array Structured Product Data at Volvo Cars	P20: Robert Bernecky: Mask and Mesh Revisited	WS3: Michael Hughes and Morten Kromberg: Windows Presentation Foundation	V03: Patrick Parks: APL+Win V10 Interpreter Performance Enhancement in Detail	Dyalog Vendor Session
10:00-11:00	P15: Martin Barghoorn: Automatic Determination of Weight for Railway Waggons	SC2: Bob Smith: APL Prototype Functions SC3: Dr. James A. Brown: The Enclose of a Simple Scalar		V04: Joe Blaze: APLNext Supervisor — A Simple Example	
11:00-11:15	*Coffee Break*				
11:15-12:15	KN2: Prof. Dr.-Ing. Horst Zuse: The Origins of the Computer				
12:15-13:15	*Lunch*				
13:15-14:15	APL2000 Vendor Forum				
14:15-14:30	*Coffee Break*				
14:30-15:30	P02: John Scholes, Jonathan Manktelow, Morten Kromberg: Unifying Traditional Functions and D-Fns in APL#	TU4: Kai Jäger: APLWiki	WS4: John Daintree: Using the Microsoft.Net Framework	V09: Klaus-Peter Friedrich: STS.win - An APL2 OLAP Database	V05: Joe Blaze: APL2000 Customer Forum
15:30-16:30	P03: Richard Smith: Processing Text Using Regular Expressions			V10: Katrin Holzmüller and Vladimir Zakgeym: APL2 at a Young Glance	V06: Joe Blaze: APLNext VisualAPL — Programming Examples
16:30-16:45	*Coffee Break*				
16:45-17:45	SC1: Volker Stamm, Bernd Stolle: How to use an APL+Win application in a .NET environment	TU5: Joe Blaze: APLNext WebServices		DPC Vendor Session	V14: Morten Kromberg: Your Application as an SQL Data Source
17:45-19:00					
19:00	Banquet				

Goto top of page schedule Monday Tuesday Wednesday Thursday

Thursday, September 16					
9:00-10:00	P06: Dr. Reiner Nussbaum: Hash arrays as Dyalog APL objects	ST1: Gitte Christensen: APL - why, when and where		V07: Joe Blaze: APLNext WebServices — A Practical Example	V15: Kai Jäger: APL2XML
10:00-11:00	P07: Roger Hui: Hashing for Tolerant Index-Of	ST2: Paul Grosvenor: Making Money with APL	GSE WG APL meeting	V08: Joe Blaze: "MVC" and "Presentation Model" System Architecture for APL	V16: Stig Nielsen: Migrating SimCorp Dimension to Dyalog APL Unicode
11:00-11:15	*Coffee Break*				
11:15-12:15	FO2: Dr. James A. Brown (Chair): The Future of Parallel Computing with APL				
12:15-13:15	*Lunch*				

13:15-14:15	DPC Vendor Forum
14:15-14:30	*Coffee Break*
14:30-15:30	Closing Session, incl. V17: Ryan Tarpine and Mstislav Elagin: Winning the Dyalog Programming Contest 2010
15:30-16:30	
16:30-16:45	*Coffee Break*
16:45-17:45	
17:45-19:00	
19:00	

Goto top of page schedule Monday Tuesday Wednesday Thursday

Conference Program APL 2010 LPA - Berlin: Summaries

Invited Paper

IP1: Dr. habil. Sven-Bodo Scholz (University of Hertfordshire, UK): Multi-/ Many-Cores: array programming at the heart of the hype!

Array programming, with its roots in data-parallelism, fits very well the programming needs of ongoing multi-/many-core development. However, there are many technical obstacles that need to be overcome to make this vision fly. We have been working on these issues over the last 15 years in the context of SaC, a functional array language (www.sac-home.org).

In this talk, I will highlight the key issues we have encountered on our journey towards compiling high-level array programs into high-performance code. In particular, I will focus on the different requirements and challenges found alongside the road when targeting diverse platforms: My journey will go from general purpose hardware, via restrained settings like GPGPUs towards cutting edge, massively parallel systems where concurrency lies at the heart of performance.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

IP2: Dr. James A. Brown (CEO, SmartArrays, Inc.): APL and its Influence on Modern Computing

This paper identifies major features of APL and its array oriented cousins and examines how they have influenced modern day computing facilities. You can find various aspects of APL available (sometimes acknowledged and sometimes not) in many offerings showing that APL has had a profound influence on computing. This is especially true if you include in the picture some of the early APL applications which could be written by people in the professions who were not professional programmers and who could produce significant applications because of the power of the notation.

Yet there are important ideas from APL that have not been exploited or are only beginning to be seen. This leads to the conclusion that APL is today a viable platform for flexible and high performing applications and that APL skills will continue to be valuable in the marketplace far into the future.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Keynote Speakers

KN1: Helmut Weber (IBM Deutschland RD GmbH): MultiCore & Hybrid Systems - New Computing Trends ?

This presentation will give an overview about recent and current developments in multicore and hybrid systems and then take a closer look at these fundamental trends in computing from multiple different angles: What are the technology drivers, what are the key system aspects and what are the programming paradigms ? And last not least - most importantly - what are the killer applications, the business drivers and the competing trends.

The discussion of these areas will outline the most important characteristics of multi-core and hybrid designs, compare them with

the more traditional design points and also similar trends in the past. Finally, the summary and conclusion will attempt to establish an outlook to the future of heterogeneous computing.

Agenda:

- Introduction:
  - Why is this interesting?
  - Recent & current developments in multi-core & hybrid systems
- Technology view:
  - Multi-core and hybrid systems as fundamental trends in computing
- System aspects:
  - Deep Integration of hardware and software with appliances
- Programming perspectives:
  - Taming Parallelism and Accelerators at the same time
- Applications and Business drivers:
  - What do we need these beasts for and what is feeding them?
- Summary and Future outlook
  - Heterogeneous computing

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

KN2: Prof. Dr.-Ing. Horst Zuse: The Origins of the Computer

Many outstanding scientists and managers were necessary to get the computer to the point of development that we know today. Konrad Zuse (1910-1995) is almost unanimously accepted as the inventor of the first working, freely programmable machine using Boolean logic and with binary floating point numbers. This Machine - called Z3 - he did finish in May 1941 in his small workshop in Berlin-Kreuzberg.

In this presentation the achievements of Charles Babbage (1823), the development of the secret COLOSSUS-Project (UK,1943), Howard Aiken's Mark I (USA), and the ENIAC (USA). Konrad Zuses contributions to computer development are presented as well, with many pictures and videos. It is not well known, that Konrad Zuse founded, in 1949, a computer company that produced 251 computers of a value of 51 Million Euros.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Reviewed Papers

Topic 1: State of the art of array processing languages

P01: Morten Kromberg, Jonathan Manktelow, John Scholes (Dyalog Ltd.): APL# - An APL for Microsoft.Net, Mono, SilverLight and MoonLight

Microsoft.Net is a software platform which is based on a component that Microsoft has named the Common Language Runtime (CLR). As the name suggests, the CLR was designed in co-operation with a diverse group of language implementers, and the services that it provides are intended to be “language agnostic”. The CLR provides an application virtual machine with memory management, exception handling and other services which significantly simplify the task of implementing new programming languages. It also facilitates communication between modules written in different languages by forcing everyone to use a single memory manager and type system.

The services provided by the CLR make it easy to implement a new programming language, and the ability to inter-operate with solutions written in other languages is very attractive to application developers. However, taking full advantage of the shared type system and related services which not only allows data to be exchanged between programming languages, but also allows the application stack to consist of functions written in different programming languages, forces us to abandon some of the most central dogma of “classic” APL interpreters:

- The notion that APL only has two data types: numbers and characters,
- That arguments are always passed “by value”, and that
- User-defined names are global by default, and local variables are visible to all sub-functions

The paper discusses the design of APL# (pronounced APL Sharp), a new dialect of APL designed with object oriented / language-agnostic platforms in mind, using Microsoft.NET as the initial target platform. Although portability of old APL code to APL# is an important consideration, the fact that complete upwards compatibility with “classic” APL is not achievable allowed us to make an attempt to “tidy up” a few other aspects of APL. The goal has been to produce a language which is as powerful a “Tool of Thought” as classic APL and APL2, at the same time as it should feel significantly more acceptable to a “software engineer”.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

P02: John Scholes, Jonathan Manktelow, Morten Kromberg (Dyalog Ltd.): Unifying Traditional Functions and D-Fns in APL#

APL systems provide a definition mechanism so that expressions may be collected into non-primitive or “user-defined” functions and operators: the *traditional* function or “T-Fn”.

In 1996, Dyalog introduced a purer *direct*-definition style, now referred to as a “D-Fn”, which was designed to fit better with the functional programming paradigm.

APL#, pronounced “APL Sharp”, is a new dialect of APL, which is aimed at the Microsoft.NET and similar “virtual machine” frameworks.

This paper details an attempt, in Dyalog's **APL#** project, to combine both “T-Fn” and “D-Fn” definition styles into a unified whole, which supports both the procedural and functional modes of programming.

The new function definition will attempt to provide the “best of both worlds” with:

- The cleanness of the D-function style.
- A vehicle for both procedural and functional programming.
- Both named and unnamed functions (and operators).
- Optional naming of arguments.
- Both traditional control structures and D-fns' guard.
- The ability to set both local and global state.

A number of other innovations are presented, including:

- An elegant approach to the definition of ambivalent functions.
- Control-structures and guards as value-returning expressions.
- The passing of nested argument “tuples”.

We hope to encourage feedback from the APL community during the specification stage of the **APL#** project.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

P03: Richard Smith (Dyalog Ltd.): Processing Text Using Regular Expressions

In so-called “scripting languages” (Perl, Ruby, Awk and Tcl, to name a few), the ability to search text using “regular expressions” is a cornerstone for the power and flexibility that these languages deliver. Although APL is (currently) mostly used to process numeric data, APL has most of the characteristics of a good scripting language, and many current and future APL applications could benefit from the availability of regular expression support tightly integrated with the language.

The support for regular expressions in Perl inspired Philip Hazel to create the Perl Compatible Regular Expression library known as PCRE, which has been incorporated into many open-source applications. Although APL vendors and tool smiths have previously implemented system or library functions which interface to PCRE and other “regex engines”, one of the typical usage patterns is to call a function to process each “match” of the regular expression within an input document, suggesting that an operator might be a more appropriate model. This paper will discuss the design decisions which led ultimately to ⎕RX, Dyalog’s first “system operator”, which can search using PCRE and make modifications to the text either by using a simple transformation syntax (similar to that used by the Unix utility “sed”), or by using an APL function to express the transformation.

The paper will illustrate some possible examples of ⎕RX in use. In doing so it will consider character classes (such as “an alphabetic character”) which may be used in regular expressions, and invite further discussion on whether there could or should be additional classes specifically included in order to support searching APL source code.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

Topic 2: Array processing languages in computer sciences

P04: Joe Blaze (APL2000): APLNext VisualAPL

APL integrated with Microsoft Visual Studio, producing fully-managed .Net Assemblies

VisualAPL is a .Net programming language which, when compiled by the Microsoft JIT compiler, emits CIL (common intermediate language) that runs under the Microsoft .Net CLR (common language runtime virtual machine) producing fully-managed .Net assemblies.

VisualAPL is integrated with Microsoft Visual Studio, so that it transparently inter-operates with any other .Net language, e.g. C# or VB.Net. VisualAPL projects may be seamlessly incorporated into a multi-programming language Visual Studio application system solution. VisualAPL adopts .Net methodology for object orientation, exception handling, debugging, index origin and object localization. VisualAPL projects may be distributed royalty-free when locally installed via an msi-format installer or when deployed using .Net "Click-Once" web-based technology.

Several unique features of VisualAPL will be discussed including:

- APLNext LAE - Lightweight Array-processing Engine
- Cielo Explorer - Interactive session for C# and VisualAPL within Visual Studio
- APLNext Scripting Engine - Create, Edit, Save, Execute and Compile APL source code scripts
- APLNext Data Serialization - Create, Edit, Save and Use XML-format representations of APL arrays
- Application-shared data store - Comprehensive support for application-scoped global objects
- Dynamic and strong data typing options of VisualAPL
- Inclusion of traditional APL assignment by value and .Net assignment by reference operators
- Enhanced execute operator as well as direct access to the Microsoft JIT compiler via compile(,) and eval() methods
- Elimination of the proprietary workspace mixing source code, application data and run-time state
- Unicode text files contain VisualAPL application system source code
- In-line C# code may be incorporated into any VisualAPL project

Also discussed will be the origins of VisualAPL, the product's target audience, the retention of core APL features including operators, functions and automatically-promoted/demoted data types, the included re-factoring tools for migrating pre-existing APL projects, some of the design decisions made for the product based on existing .Net features and security requirements and techniques for comparative performance measurement of VisualAPL projects.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P05: Joe Blaze (APL2000): APL+Win Interfaces

APL+Win has been, and will continue to be, enhanced numerous times to interface with popular ‘external’ objects. These APL+Win interfaces include COM/ActiveX support, Microsoft .Net assemblies, Microsoft .Net ADO databases, email, XML-format text, zip-format archives, native files, APL component files, the Windows command prompt, Unicode-based information, source code repositories, pre-.Net dll’s and assembly language.

APL+Win can access any ActiveX (COM) component:

- Automate Microsoft Office: Word, Excel, Powerpoint, Access, Outlook, etc.
- APLGrid, a royalty-free spread sheet integrated with APL+Win product
- Any .Net assembly that is exposed as ActiveX (COM).

APL+Win can access any .Net Assembly:

- Use the Lescasse NetAccess tool, included with the APL+Win product to expose any .Net assembly as COM and use it from APL+Win.
- Create .Net user controls and deploy them in an APL+Win-based GUI
- Use the .Net LINQ tools to manipulate XML or Microsoft SQL Server data in APL+Win application systems
- Create custom .Net assemblies and use them in APL+Win

APL+Win can access the ADO.Net database tools:

- The APLNext Database Interface Tools for Microsoft SQL Server, Oracle and IBM DB2

APL+Win can use CDO, the Microsoft Microsoft ActiveX (COM) application programming interface to compose and send email

APL+Win can read, write, edit and manipulate XML documents:

- The Microsoft ActiveX (COM) XML DOM (document object model)
- The XML Tools workspace which implements XML as APL nested arrays, available at http://forum.apl2000.com/viewtopic.php?t=43

APL+Win can read, write and create .zip archives

- The APL+Win □dr system function supports deflate/inflate of .zip archives.

APL+Win can read, write, create and delete native files

- The □n- and □xn- system functions provide this functionality

APL+Win can read, write, create and delete traditional APL component files

- The □f-, □xf-, and □cf- system functions provide this functionality for legacy 8.3, extended and colossal APL component files respectively.

APL+Win can run DOS commands directly

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P06: Dr. Reiner Nussbaum: Hash arrays as Dyalog APL objects

This paper presents a method to store arbitrary key / value parsing a hash array. Usually, data is stored in data arrays, where space is available for each possible element for the array within given limits. An element is looked up by retrieving the data element by indexing the data area. The talk describes a solution of hash arrays implemented as a Dyalog APL object which covers the main functionality of perl's hash arrays.

Hash arrays only contain information about data elements, which are present in the array. A hash array may be seen as a pair of vectors, containing key and value information. Hash arrays are useful in the case, when in a large logical array only few values are present. Such arrays usually are called sparse arrays. Besides of the sparse arrays problem, hash arrays can be used to handle a general key / value relationship of any (APL-)object. So, using hash arrays, it is not a problem to have a picture as a key and a video stream as a value.

The possibility to specify a hash function to process hash keys during the object's initialization phase offers a method to customize the hash array and optimize performance relating item processing. The hash function is given to the APL object in the canonical form of a Dyalog APL self defined function.

Of course, insert, remove and modify member functions to deal with a key/value pair are present. Thru the nature of APL, applying the each operator, a bulk processing of many key/value pairs is possible without having any additional coding. Read operations generally could be executed in parallel.

On top of the basic member functions mentioned above, more members are implemented in the APL object. Three of them are mentioned here. The retrieve keys and retrieve values member functions are useful, if special processing is to be performed under user control. Using only the retrieve keys member function with respect to sparse arrays it easily is possible to select the keys for a special plane in a cube and next retrieve all relevant data in one step. Another interesting member function is the swap member which interchanges the key / value relationship. In cases of unique data within a hash array, then it is immediately possible to find the key value for a given data element.

The talk covers the description of the concepts and the implementation and a live demo as well.

Interested? You're welcome to join the talk.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P07: Roger Hui: Hashing for Tolerant Index-Of

We consider the problem of x⍳y where x and y are real (64-bit floats) or complex (64-bit floats of the real and imaginary parts), with tolerant comparison (nonzero □ct). We show how to use hashing to solve the problem.

Hashing algorithms for real and complex arguments have been implemented in Dyalog v13.0. APL models and test cases are provided. Benchmarks demonstrate the improvement over the previous implementation.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Topic 3: Parallel programming with array processing languages

P08: Devon McCormick: Parallel Programming Theory and Examples Towards Sketching a Taxonomy for Problem Estimation

We begin with a brief overview of the history of parallel processing ideas and terms. We will cover some of the frameworks in which these ideas have been organized with particular emphasis on the distinction between fine- and coarse-grained parallelism. We will also bring up the conflict between the specific limitations of particular parallel hardware architectures versus the importance of abstraction and generality sought for software.

Next, we will look at three of the current, dominant varieties of parallelism: distributed computing, graphical processing units, and multi-core chips. In discussing the strengths and weaknesses of each of these parallelization targets, we will start to develop some

of the major elements comprising a taxonomy under which we can classify different types of parallel computing solutions. We will begin to elaborate on the distinguishing features to consider when evaluating approaches to parallelizing an algorithm or specific problem solution.

We will continue by considering how some of these major features of parallelization apply to array-processing languages (APLs) and how this gives rise to a dichotomy between what support we would like to have implicit in an APL to take advantage of some of the strengths of this paradigm versus more immediate, practical considerations. To demonstrate this latter consideration, we will look at a number of specific examples of attempts, with varying degrees of success, to parallelize some applications to take advantage of one of the most readily available forms of parallelism: the multi-core chip.

In this penultimate section, we will consider a hierarchy of increasingly difficult problems - examining in detail at least one success and one failure - in attempts to speed up processes through multi-core parallelism. As part of this we will also briefly touch on some of the pitfalls of comparing different implementations and offer some tips on tools for measure different aspects of system performance. More importantly, we will also illustrate, with complete examples of code, some of the potential penalties incurred by the increased overhead and greater complexity of a parallel version of an implementation.

Finally, we will summarize the major considerations for implementing parallel processing to at least begin to establish a taxonomy of parallel processing problem conversion. This taxonomy should be general enough to apply to a wide range of problems yet be sufficiently specific to give useful guidance on how to approach a particular task.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P09: Joe Blaze (APL2000): APLNext Supervisor

The APLNextSupervisor is a productivity platform for executing APL+Win functions in a multi-threaded manner. When the computer hardware on which APL+Win has been installed has multiple processors or multiple ‘cores’, installing the APLNextSupervisor and configuring the APL+Win application system to use it means that programmer-designated operations associated with APL+Win functions can be executed concurrently. This allows the developer to take advantage of modern multi-core hardware.

Concurrent execution of an APL+Win function means that the processing performed by that function occurs asynchronously with respect to other processes of the APL+Win application system. Thus the ‘main thread’ of an APL+Win application system is not ‘blocked’ while an APL+Win function of that application system is running concurrently because it is running in a separate ‘thread’. Results from these multiple threads are returned to the “controlling” application via events which are fired when a thread completes execution.

Application systems which can identify sections of the processing algorithm that can be asynchronously executed can usually benefit from multi-threading. Typically such application systems perform repeated, in-memory processing of a collection of similar data elements, such as those associated with time series, populations, stochastic processes, simulations, etc.

Why Multi-Threading

How is Parallel Processing Implemented

Configuring an APL+Win Application System for Parallel Processing

APLNext Supervisor FAQ

APLNext Supervisor: Methods, Properties and Events

APLNext Supervisor Configuration Schema

Examples: Stochastic simulation of time series, Valuation of Employee Benefits

The APLNextSupervisor can be accessed from any .Net or Win32 programming language

- APL+Win Source Code Example
- C# Source Code Example

Distributing the APLNext Supervisor with your application system

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P10: Wai-Mee Ching, Da Zheng (Zhejiang Normal University, Jinhua, Zhejiang, China and Johns Hopkins University, Baltimore, Maryland, USA): Benefits and Limitations of Array-style Programs for Parallel Execution

We recently implemented a parallelizing APL compiler on an Intel architecture 4-core desktop computer running Linux. The compiler is based on an APL-to-C translator with an added enhancement to insert OpenMP directives at appropriate places in the generated C code for parallel execution. We measured the runtime and speedups on several programs written in classical APL-style, i.e. array-oriented, and observed quite good speedups in all these examples, particularly the speedups achieved during execution of array operations such as inner and outer products. This leads us to a discussion of the benefits and limitations of array-style programming for the parallel execution on modern shared-memory machines, or more specifically current multi-core desktop machines.

The first benefit is that the parallel speedup comes at no extra burden to an APL user (this would also be true for MATLAB users if a MATLAB compiler has already been similarly implemented) other than that he needs to compile his program: there is no variable declarations, no data-layout declarations as in HPF, and no hints to indicate places in a program for parallelization opportunities. In any case, there is no need to learn a new parallel programming model as APL's semantics is neither sequential nor parallel but mathematical. The second benefit is the simplicity of providing automatic parallelization: unlike conventional scalar language based automatic parallelization systems, there is no need to carefully reconstruct potential parallel operations from extensive analysis of loops when the source program is array-based.

There are also limitations to achieve parallelism solely using array-oriented programming. The first is that a naive implementation of an array-style program can easily incur a large amount of unnecessary computations and data movement as we have seen in our work mentioned above. This limitation may be overcome by sophisticated compiler implementation. The second limitation is more structural and challenging: there are many computation intensive applications which are either of irregular structure or iterative in nature (such as Monte Carlo simulation). A first step to overcome that limitation is to extend our compiler work to implement task parallelism, not just data parallelism as we did so far. The next step will be to introduce some innovative data structure, not a mere general array, to indicate the pattern of application parallelism, coupled with new compiler work.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P11: Morten Kromberg, Michael Hughes (Dyalog Ltd.): Parallel Computation Using Peach, Prank and Pouter

One of the challenges currently facing software developers is to take advantage of the parallel hardware that is appearing not only in large data centers, but also on every desktop. APL is an inherently parallel notation, which has the potential to make this relatively easy. At the lowest level, users should be able to expect the APL interpreter to distribute computations optimally across multiple cores, when evaluating expressions like:

```
1 2 3 + 4 5 6 × 7 8 9
```

In fact, achieving this is not as simple as it seems, as there are many bottlenecks in multi-processor machines which mean that efficient use of multiple cores by a "SIMD Interpreter" is a significant challenge. Research also shows that the average number of elements in arrays passed to primitive functions in commercial applications is less than 2, which suggests that parallelism at the level of individual primitive functions is unlikely to help typical applications very much.

At a slightly higher level, APL implementations offer a number of parallel constructs:

- The each operator (¨), which is available in virtually all modern systems
- The rank operator (⍤), implemented in SHARP APL and J (and often emulated using user-defined operators in other APL systems)
- The outer product operator (∘.), which is available in all systems
- The dot in Dyalog APL: When placed between an array of objects on the left and an expression on the right, dot applies the expression to its right to each of the objects on the left.

Each, rank outer product and dot are different ways to express the application of primitive, derived or user-defined functions in parallel. Unfortunately, because user-defined functions (and some system functions) can have side-effects, APL interpreters cannot know whether it is safe to execute the multiple calls to the function f in expressions like (f¨data), (f⍤1 data) or (objects.f data) in parallel. The paper describes four experimental user-defined operators named Peach, Prank, Pouter and Pdot, each of which allows the user to explicitly declare that an expression is parallelizable.

These models of potential extensions to an APL interpreter are implemented by forking multiple processes which communicate using TCP/IP, and allow us to experiment with the performance characteristics of parallel execution using multiple cores in one or more co-operating machines, and the tuning parameters that may be required to optimize throughput on different hardware configurations. So far, experiments suggest that, although they may require small changes to application code, the use of these operators has the potential to provide significantly more "bang for the buck" than the implementation of fine-grained parallelization in interpreters.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Topic 4: Challenging (production type) applications solved with array processing languages

P12: Helmut Engelke: Improving Violinists' Intonation

Pure intonation calls for a much larger gamut than that provided by a standard piano. In this paper the implications for string players are studied.

Singers are free to intonate a melody according to their musical ear and personal taste. Pianists depend on fixed preparations of their instrument including its musical temperament. A violinist finds himself in an intermediate position. After tuning his four strings he can intonate his music as he likes. If he plays in an ensemble, he should adapt to less flexible instruments and also compromise with co-players. In this paper we focus on solo playing and restrict intonation on pitch.

Harmonic tonal systems and violin strings are analyzed to help violinists to a better intonation.

In a first step, the *mathematical* properties of known harmonic tonal systems are revisited using the *Euler space* of intervals. The interval spectrum for a suitable subspace is derived and visualized in elaborate diagrams. Audio samples of intervals and scales will also be presented.

In a second step the physical properties of a stretched and oscillating string are summarized. The effect of tension on intonation is demonstrated in an experiment. Some musically interesting string properties are studied by taking two or more instances of the related Eigen frequency equation. By combining interval spectra with string properties virtual fingerboard frets can be calculated.

Practical recommendations for violinists are formulated and compared with those of prominent violin pedagogues —past and present. Numerical results were achieved and visualized using APL2 and the graphic auxiliary processor AP207. AP207 again proved to be an excellent tool for rapid prototyping.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P13: Markos Mitsos (DKV): Building reference systems in German health insurance

The calculation of premiums in German health insurance is based on two pillars:

- 3 fundamental actuarial tables, each depending on gender and age, namely mortality (qx), lapse (wx) and claims per capita and year (expected health costs, Kx), combined with an expected long term average interest (technical interest rate i).
- The premise that the individual premium shall not (barring a change in the underlying actuarial assumptions) depend on aging itself, but only on the age at contract date.

As a result (and as easily verified by some numbers and a very brief overview) the business model of German health insurance is build around (huge) benefit reserves. On the other hand German health insurance is very tightly regulated, and subsequent new laws have complicated the calculation of premiums and the accounting standards for reserves ever more.

For these reasons the benefit reserve is the central item in the balance sheet. Similarly the annual compulsory control of the actuarial assumptions, regularly resulting in premium recalculations, is the central business process in German health insurance. The calculation of the reserves as well as the recalculation of the individual premiums based on the control mechanism, are straightforward. However they consist of a very large number of simple arithmetic operations and incorporate many small differences for all the plans in the portfolio. Reference systems for both are therefore paramount.

There are some technical and actuarial prerequisites for the implementation of a fast and reliable reference system in APL. Among other things some basic algorithms for hardcore data processing (like grouped sums of large arrays) and functions for communicating with databases, Excel and other COM-objects. Those are copied from other workspaces and taken for granted. On the actuarial level higher level actuarial assumptions (annuity tables, commutation values etc.) in a form suitable to APL are needed. In our case the programs generating those are common to all reference systems as well as other programs and not discussed here. On the technical level new operative programs are regularly tested, and the testing environment must conform to actuarial standards. The testing environment for the presented reference systems is also maintained in large part by the actuarial department through APL+Win and with the help of DB2 utilities. Its existence is just accepted here.

The basic schema of the control is relative simple and straightforward - as the logic behind the referenced systems itself dictates. Because the primary data and the results lie in databases on the mainframe, on which the access through individual data processing underlies restrictions (workload manager etc.) the are read just once, brought into an APL-optimized array form and kept on the PC side and in component files. One central APL+Win function does all the computations that are to be verified, with an emphasis on not using -as far as possible- similar algorithms, in order to avoid basic thinking errors. This gives the extra benefit of a concise and hopefully equivalent formula for all the nested and overly complicated computations done in legacy mainframe COBOL. Usually, and as will be explained in detail, two results are produced, exposing different kinds of problems.

Before comparing the three results (original and two simulations) an ensemble of cases containing "every interesting constellation" is chosen. These special cases are compared and the results (regardless of differences) presented in great detail in a series of Excel Workbooks. Then a certain closure (not an algebraic or topological one!) of the special cases is taken and the results compared. An overview lands in Excel, containing only cases with problems and a listing of numbers of mismatches and maximal absolute differences for each plan. Finally all cases are compared and the resulting differences, accompanied by a listing

of summed differences for each plan, are also exported to Excel. The different outputs serve different purposes, as shall be explained.

The main implementation difficulties lie in consequently enforcing four byte integer as the predominant data type, especially for the main data arrays to enhance performance and save memory; abstracting existing calculations to schemata compatible with matrix manipulation and containing as few exceptions as possible; keeping the simulation minimalistic, simple and "open minded", allowing greater flexibility when reacting to a changing environment. The latter is important in a corporation with two large and old companies which have acquired three smaller ones: Some special processes in very old plans (possibly including the thinking errors mentioned above) are so deeply ingrained in the operative systems that they cannot be questioned. Approaching the same plans with the simplest actuarial method suitable exposes some differences. It may be difficult, or at least time consuming, to decide what is really correct and ensure its implementation. In the mean time it must be easy to implement some kind of switch in the simulation, to separate the open cases from other possible errors.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P14: Lars Wentzel (Fujitsu Sweden): CPAM — Array Structured Product Data at Volvo Cars

The Cars from Volvo are extremely diversified products. They can be configured in billions of different variants. The reason is that Volvo has a broad international market with different needs and wishes in different countries as well as amongst individual customers.

The product information is used throughout the company, amongst partners and by customers. You need this in public product information, sales, ordering, planning, logistics, production, distribution, procurement, product development and finance. For efficient company operations it is vital that this data is easy to reach, correct and up-to-date. The problem becomes even greater when you realize that the cars change frequently so that you have to describe in detail when all the changes will take place.

The best way to accomplish the information needs is to create a common master system with this data where you can reach it by services e.g. web-services (SOA). During a ten years period we have created this and we did it with APL i.e. Dyalog APL for Windows.

The first task was to create regular translation of the complex rules of the legacy Engineering System. We did this by using some of the principles described by the Danish company Array Technology. So we take data with sequential rules, investigate all the possible combinations and create array structures that in a compact form describe all possible combinations. Such an array structure is set of several hundred connected (nested) arrays. When you have this data in array format the revolution starts. You can now use it for multiple purposes like creating new forms of presentation and to analyze data from different angles and with very good performance.

After this we created the data engine to supply other systems and users with services. This is accomplished by using a set of continually running Dyalog APL server sessions. The most frequently used product information is held as variables in working memory. The sessions are all waiting for web-service request and then create the reply. This is all in the form of web-services. There are more than 100 different services for external or internal use. The web services are all done using the build in functions for TCP/IP in Dyalog APL. Recently we have started using the Conga interface. The XML parser functions etc. are our own make.

In addition to this CPAM contains

- Interactive analysis and updating with a browser interface using IIS, ASP, HTML etc. communicating with the Dyalog APL server engine.
- Translation services where CPAM actively sent product information to other systems
- Batch services where you get a bulk of data, processes it as a batch and then return it
- Ordering of Excel reports

CPAM currently supply 800 users with product information and interacts with around 20 systems.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P15: Martin Barghoorn (TU-Berlin): Automatic Determination of Weight for Railway Waggons

Dynamic Wheel Force Measurement
For five years we have been developing an APL-program to obtain the weight of waggons from a moving train. This method is called dynamic wheel force measurement, DWFM. DWFM is a quick and effective way to dynamically measure the weight of moving trains. It makes time-consuming and expensive static weighing of fully detached waggons on costy static balance scales obsolete.

DWFM is a fully integrated system consisting of strain gauges, a signal converter box and a laptop. It can easily be moved and

applied to any railway track and therefore is highly flexible in use.

The strain gauges are installed at specific positions on the side of the rail track. The strain gauges are supplied with voltage from a power source. The elastic deformation of the track steel when a train wheel is passing is indicated by the change in electrical resistance of the strain gauge circuit.

The measurement is amplified and transmitted via TCPIP interface to a computer harddisc txt-file. In correlation to the amount of waggons/locomotives, the size of this file can vary from 1 to 10 MBytes. An APL2-software reads the data sequentially via the associated processor 12 (files as arrays) from the external txt-file from both channels which are caused by the left and right track. After controlling the data the calculation of statistical parameters (min, max, mean, median etc.) is carried out.

Automatic detection of waggons
The physical equation $d = v \cdot t$ relates to distance, velocity and time. As the speed of the train is $v = constant$, the relation of d and t is directly proportional, $d \sim t$. Therefore, the distance between two axis can be determinded with the time. With the help of expert knowledge, the waggon can be distinguished by type (2-axis, 3-axis, 4- axis, 6-axis etc).

Results are written in a database
The processed data is prepared and the results are stored in a database. The relevant output is shown directly via dot.com in an excel-sheet.

Java as an interface for APL
The control of the software is done by a user friendly Java-program.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Topic 5: Theory and practice in array processing language design and implementation

P16: Patrick Parks (APL2000): New APL+Win System Features

The latest release of APL+Win introduces several new features designed to make applications easier to code, easier to debug, more reliable, and easier for APL+Win system developers to diagnose and fix catastrophic system failures (crashes) that occur in deployed applications at remote client sites.

Inline control sequences can be coded anywhere an APL expression can be used. They provide progressive partial evaluation similar to the &&, ||, and ? operators in C, C++, Java, etc. For example:

```
:IF (A Fu B :OR C Gu D :OR E Hu F) :AND (G Mu H :OR I Nu J)
  Do Something Interesting Here
:ENDIF

Foo (A > B :THEN A Goo B :ELSE B Hoo C)

Foo (X :CHOOSE Goo W : Hoo X : Moo Y : Noo Z :ELSE Other)
```

In the first example, if (A Fu B) is true (C Gu D) and (E Hu F) are NOT evaluated because we already know the result of the :OR sequence will be true. Similarly, if (G Mu H) is true we don't need to evaluate (I Nu J). On the other hand, if (A Fu B) and (C Gu D) and (E Hu F) are all false, then we don't need to evaluate (G Mu H) or (I Nu J) to know the overall result is false. The :THEN keyword provides a binary choice between two execution alternatives, only one of which is executed. The :CHOOSE keyword provides an indexed selection among multiple choices, only one of which is executed.

The :TRY statement has been enhanced in four ways. A localized error handler can be applied specifically to the context of the try block. A shorthand * notation can be coded to automatically bubble up all errors that occur at any depth of function nesting back to the :TRY context for handling. A :CATCH statement has been added that uses a wildcard pattern matching argument to select which errors it traps. This is simpler to code and more efficient than using APL expression tests. The :FINALLY clause always executes no matter how control leaves the :TRY or :CATCH statement contexts (by error, return, branching, or other means). This provides a sure fire way of releasing resources such as closing files or database connections when leaving the controlled context. A new debugging mode can stop for errors at their point of origin to help in debugging errors in :TRY blocks.

The :ASSERT statement tests an expression and causes an assertion failure when not true. It can be disabled in deployed applications in which case it has zero execution overhead. But it can be re-enabled in the field via configuration file to help diagnose problems that arise after deployment. Similarly, the :DEBUG statement doesn't execute in deployed mode but can be re-enabled via configuration file.

Trace logging can be enabled to capture execution history for up to 134 million statements. This history can be accessed programmatically or dumped to a file when the application exits or the system crashes. This can help diagnose application logic errors and can be enabled in the field via configuration file. It can also be useful for helping APLNow diagnose APL+Win system errors in deployed applications.

Historically, when APL+Win crashed due to system errors or errors in 3rd party components, it didn't leave a trace to help APLNow developers diagnose what might have caused the problem. Catastrophic error handling has been completely overhauled so that if and when a crash occurs the application can restart itself to recover from the error. In addition the system now captures as much diagnostically useful information as possible include log files and MiniDump files for remote post-mortem debugging.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P17: Dr. Herman Singer (Syndeon Soft): Succinct - A new APL dialect

One of APL's strongest points since its introduction nearly 45 years ago, and probably the single most distinguishing feature compared to mainstream languages such as C or Java, is the ease of transition between 1:1, n:1, 1:n and n:n multiplicities without having to reformulate code. This makes it an interesting choice for algorithm prototyping, algorithm testing and mathematical array processing. Therefore APLs are regularly used in mathematical and statistical applications such as actuarial research or financial transaction analysis. Traditional APL implementations however seem to lag behind recent developments in networking, Internet applications and string handling. While there are some vendor specific solutions to provide some functionalities by additional libraries this kind of processing seems to remain somewhat foreign to APL.

Succinct is a new APL-like general purpose programming language, which introduces advanced programming concepts such as closures, generators, lightweight threads, and co-routines as well as fundamental integration of regular expressions for string handling and distributed/network programming, and implements all essential APL functions and operators. Succinct encourages functional and/or object oriented design with hierarchical name spaces, objects, inheritance and delegates. Functional composition is augmented by allowing tacit programming.

Unlike traditional APL Succinct uses lists and not strict arrays as the fundamental composite data type. This enables Succinct to form trees naturally without the need to pack and unpack them as normally done in APL. Therefore algorithms or data structures involving trees can be easily implemented and manipulated.

Control structures of traditional languages (C and derivatives) such as different conditional and loop constructs are present in Succinct so that sequential algorithms can be easily transcribed and then as necessary or possible improved into array processing expressions.

Succinct can be used in the traditional way within a graphical workbench and interactive environment which is part of the distribution, or as scripts in the standard UNIX-scripting way.

Syntactically Succinct uses only ASCII-characters and is thus similar to J or K. In fact many operators have the same or very similar meaning as in K, so users acquainted with K/Q should not have problems to read Succinct code.

Succinct is platform independent and runs on Windows, Linux and Mac OS X and is thus well suited for cross platform applications. In particular Succinct is able to produce standalone executables without the need to redistribute the run time environment separately.

The small size of the Succinct interpreter and its reasonably fast performance make it a competitive candidate for applications where traditionally other languages such as Perl, Python or Ruby are used. The tight integration of string processing, in particular with regular expressions on the language level allow for tasks of system administration and server-side processing in addition to the standard mathematical prototyping capabilities.

Integration of communication with ODBC SQL databases and SQLite make Succinct useful for database applications. Bindings to OpenGL as well as possibilities to create GUI programs enhance its usability.

Demonstrations of Succinct's capabilities of string processing with regular expressions, distributed computing, graphical interfaces and OpenGL will be given.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P18: Richard Smith (Dyalog Ltd.): Damage Resistant Component Files Using Journaling and Other Techniques

Dyalog component files are maintained by the interpreter using indices and other high-level structures within an underlying "native" file, which is often shared by many users on multi-user operating systems and networks. Each component file update (e.g ⎕FAPPEND or ⎕FREPLACE) requires a number of updates to different parts of this underlying file; once started, and until the process is complete, the file may be in an inconsistent state. The series of updates must appear to be atomic to other users and locks are used to ensure that files are not accessed by other users when the file state is inconsistent.

Component files have worked in this way — and generally served APL well — for many years. But many APL users have also experienced damaged component files — ones which are impossible to read or update, which crash the interpreter or, worst of all, present no ill effects but return corrupt values. Almost always the cause of such damage will have been the interruption of a file

update (perhaps due to a kill signal to the application making the update, or a network failure), leaving the file state inconsistent.

Dyalog APL has addressed this problem in two stages, resulting in component files which are highly robust and resilient to damage. The first stage protected the file from damage caused by interrupting an application during an update, and introduced journaling to the process. The second stage dealt with the damage which results from the loss of file caches following a catastrophic event such as a power failure or operating system crash during an update, and extended journaling to include cache flushing and checksums.

In addition to the technical details of component file structure, the critical points during an update sequence and the methods used by Dyalog to protect and repair files, this paper will also discuss migration of existing files and compatibility between Dyalog releases, and consider the benefits of journaling versus the additional processing involved - so that the best journaling options may be selected for any particular application.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P19: Geoff Streeter (Dyalog Ltd.): Supporting APL keyboards on Linux

Xkb is a keyboard extension for X windows. Linux provides a full implementation of the Xkb extension. It has been used to provide a clean and very usable APL keyboard.

The APL special characters are supplied by this keyboard over and above the characters supplied by the user's normal choice of keyboard.

The work builds on that of ISO9995 and Erik Fortune of Silicon Graphics. It enables a “latching” (active whilst pressed) shift key to access the APL characters without conflict with other uses of the keyboard.

The APL characters are available to any application running under an X server including; word processors, email, terminal emulators, shells …

The X client can be a program running either on the same machine as the X server or on a remote machine anywhere in the world.This keyboard is not Dyalog specific it can be used by any APL, or indeed, non-APL program. However, Dyalog have produced, fully engraved, APL keyboards for US, UK, and Danish users. The APL keyboard produced for Xkb provides the support necessary to use those additional engraved characters in the positions that Dyalog have chosen.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

P20: Robert Bernecky (Snake Island Research Inc.): Mask and Mesh Revisited

Kenneth E. Iverson described the `mesh` and `mask` primitives in *A Programming Language*, nearly forty years ago, yet no array language has implemented them as primitives. The array language SAC has, however, included `mask` as a standard library function, `where`, and Dyalog APL provides a related D-function, which they unfortunately call `mesh`.

Both functions are simple and straightforward, compared to equivalent APL expressions:

Mesh:

```
    'sek' 0 1 0 1 0 \ 'ta'
steak
```

Mask:

```
    'abcde' 0 1 0 1 0 / 'ABCDE'
aBcDe
```

This `mask` example can be written using conventional APL as:

```
b←0 1 0 1 0 ⋄ i←⍸b/⍳⍴b ⋄ z←'abcde' ⋄ z[i]←⍸b/'ABCDE'
```

In a conventional APL interpreter, this functionality would require about a dozen memory management operations, and run-time overhead to analyze and dispatch five primitive operations. With Iverson's `mesh` and `mask`, one of each does the job; they also make programs simpler and more readable.

We have modified the SAC array language compiler to perform *algebraic with-loop folding* on arrays of arbitrary shape and known rank, an extension of Scholz's *with-loop-folding* array optimization. These optimizations are powerful enough to transform the standard library `where` function into a single data-parallel with-loop. In common cases, that loop will also contain the expression that computes the control argument, `b` in the above example, thereby avoiding creation of an array-valued temporary. We report on the performance of SAC-based mesh and mask benchmarks, compared to that of their APL equivalents, including serial speedup, parallel speedup, cache miss rates, and so forth.

Reviewed Short Communications

Topic 4: Challenging (production type) applications solved with array processing languages

SC1: Volker Stamm, Bernd Stolle (a.k.e Software GmbH): How to use an APL+Win application in a .NET environment

We want to introduce a way to use an APL+Win application in a .NET environment. This is necessary if an APL+Win application is needed within a .NET environment but cannot be rewritten to an e.g. VisualAPL application, because it is also running outside the .NET world. Before we show how to use the APL+Win application we take a closer look to the concept of middleware and particularly .NET.

Middleware is a concept in modern software architecture and of growing importance. It offers better possibilities of interoperability of single applications, improves ways of scaling systems and last but not least lowers time and costs of development.

Usually middleware products offer a big variety of tools to make development more comfortable. A central idea of middleware is modularisation at application level. Thus well defined interfaces on application level become more important. This enables quick and easy reuse of applications in other contexts and so offers the potential of lowering time and costs of development.

Two important examples of middleware are Java EE and .NET. Both are designed to offer independence from OS platform, but so far full .NET is available on Windows only. On the other hand .Net supports a wide range of languages while Java EE is (more or less) restricted to Java. “mono” and “GNUnet” are examples for implementations of .NET for other OS platforms.

Let's take a closer look to Microsoft's .NET and its IDE: Visual Studio 2008. The IDE Visual Studio 2008 is running on Windows only. Applications for .NET are compiled to the so-called common intermediate language (CIL) instead of machine code. They run on a virtual machine called common language runtime (CLR). .NET distinguishes between managed and unmanaged code. Managed code is any code running on the CLR and unmanaged code is any other running application. .NET with Visual Studio 2008 can use COM/ActiveX applications.

APL+Win is not a .NET language. Nevertheless there is a feasible way to connect APL+Win to the .NET environment: the ActiveX-Server. ActiveX is supported as well by .NET on Windows as by APL+Win. We recommend an XML formatted string to exchange Data. This will help to avoid the technical differences of data types between ActiveX and the .NET environment and to maintain the idea of reusability. Using an XML formatted string also enables an exchange of complex data structures. An auxiliary feature of XML is the pretty simple definition of this standard and its wide adoption, so there exist many (free) tools to read and write XML. This reduces additional effort to add an XML interface to the APL+Win application.

In conclusion we obtain a quite convenient way of using an APL+Win application in a .NET environment by combining the usage of the ActiveX server of APL+Win and XML formatted strings for exchange of data.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

Topic 5: Theory and practice in array processing language design and implementation

SC2: Bob Smith (Sudley Place Software): APL Prototype Functions

Abstract
The concept of empty arrays while not unique to APL, has been most thoroughly developed there. Nonetheless, the rules for handling empty arrays are inconsistent across the major APL implementations where, for example, the sum of two empty vectors can not only give different results, but also, sometimes (and correctly so) a LENGTH, RANK or DOMAIN ERROR. Moreover, on certain implementations addition of empty arrays is not commutative in general (e.g., L+R does not match R+L).

This paper attempts to give a comprehensive treatment of prototypes and set down consistent rules for computations with them in order to encourage all APL implementations to produce the same results. Along the way, we introduce the concept of a prototype function - a function associated with each primitive and derived function to be used in lieu of the original function when applied to empty argument(s).

Rationale
From the very beginning, APL designers and implementors have prided themselves on how the primitive functions worked “as expected” in the limiting empty case - that is, on empty arrays. Few if any other languages take such care to handle such edge conditions, largely because empty arrays play such a small role elsewhere. In APL, empty arrays are huge!

Programmers don't normally start out creating empty arrays, they just happen in the course of normal (or abnormal) computation. This is why developers need to expend the effort to ensure that their implementation handles them seamlessly. By treating empty arrays as a natural and limiting case of data, developers can shift the burden from programmers so the latter can concentrate on algorithms, not edge conditions. More than in most languages, it is the APL developers' responsibility to simplify the programmers job, part of which is to get the edge conditions right. Effort in support of this goal can reduce programmers' frustration as they wonder why a particular piece of code doesn't seem to work, or works differently from vendor to vendor.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

SC3: Dr. James A. Brown (CEO, SmartArrays, Inc.): The Enclose of a Simple Scalar

As original APL evolved to APL with nested arrays, probably no issue was more divisive than the topic of enclose of a simple scalar since it was a fundamental issue of the nature of arrays. This paper shows that the choice made by IBM APL2 is reasonable by developing and utilizing some algebra for determining if two arrays are the same array. A "“Structure"” meta-function is defined for a uniform subset of arrays. It is then postulated that if two arrays have the same structure and the same values at each position, then they are the same array. This analysis is applied to a simple scalar and then to the enclose of a simple scalar.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Tutorials

TU1: Bernd Geisselhardt (Allianz Deutschland): Development Environment on the Workstation and Runtime Environment on the HOST? It works!

A short overview will be given over the hardware/software configuration at Allianz including DB2 program database.

The APL application development process and environment on both platforms will be explained, in particular the Allianz production release authorization.

The whole process will be live demonstrated on the Allianz mainframe environment

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

TU2: Patrick Parks (APL2000): APL+Win Performance

The latest release of APL+Win introduces dramatic improvements in workspace size and execution speed. Taking full advantage of these requires understanding how they work in detail. This session will discuss performance characteristics and tuning of applications. Some of the principles discussed may be of interest to users of other APL system. But the focus will be on giving guidelines for achieving optimal performance that is tuned to the specific characteristics of their application running on APL+Win. We will often dig down into fine grained ”APL+Win insider’s tour” details about how the system is implemented in order to give sufficient insight into why these guidelines make sense.

Some users who have tested the APL+Win 10.0 system report that their applications run in ½ the time they took on previous versions. They don’t see as much improvement in I/O intensive applications that spend moving large amounts of data around on disk or across the internet. But calculation intensive applications that “crunch” a lot of data in APL as well as applications with a lot of decisional and iterative logic generally run must faster. Some of this improvement has been achieved by reversing slowdowns that crept into the system as new features were added over the years. But the performance we are seeing now is considerably faster than any previous release of APL+Win. We will highlight exactly where performance has improved so that users have a better idea if their applications might benefit.

Workspace size has also increased dramatically from a maximum of about 1.7 GB on previous versions up to about 3.7 GB (on Win64 operating systems) or 2.7 GB (on Win32 operating system with the 4GT option enabled). This was done while remaining a 32-bit application which means most existing ActiveX and DLL components as well as workspace based Assembler Functions (QuadCALL) used existing applications will continue to work without modification. If we had achieved this size increase by porting APL+Win to be a 64-bit application, such components would have needed replacement by 64-bit versions of those tools (if they are even available). But a 64-bit version may soon be under way as well.

As workspace sizes increase it becomes more important to understand the huge impact workspace memory management can have on overall performance of an application. In most applications, the overhead of allocating, freeing, recycling, and reorganizing workspace memory can account for a greater fraction of time than is spent on actually “crunching” numbers. While this is all handled automatically “behind the curtains” for APL applications, there are decisions developers can make and actions they can

take that have a dramatic impact on this aspect of performance, either positively or negatively.

A great deal of workspace memory management tuning is handled automatically and adaptively by the APL+Win system as it learns how an application behaves and changes memory management strategies accordingly. But there are hints that the developer can give to the system that enable it to make fewer guesses and better decisions. Hence, it is possible for the astute developer to help improve performance if they are given the necessary tools and insights and are motivated to spend a little extra time to optimize their application with respect to workspace memory usage.

Performance tuning cannot be considered in a vacuum that looks only at how one instance of an APL+Win application performs in isolation. Balance is important and well done applications need to be considerate of other applications running at the same time on the same computer. Therefore, we will also be discussing how to be a good neighbor to other applications. Even from a selfish perspective, when running multiple instances of APL to work on different parts of a problem in parallel, being a good neighbor to your own application instances can improve the overall throughput of the system.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

TU3: Dan Baronet (Dyalog Ltd.): User Commands in Dyalog APL

User Commands (UCMDs) are similar to system commands, except that they are written in APL. They provide a way to make tools and utilities available at all times — without requiring them to be copied into the active workspace before use. The user command processor also provides a mechanism for on line help, and encourages developers of UCMDs to provide consistent behavior across different commands.

Dyalog introduced User Commands with version 12.1. In the Dyalog implementation, the source for a UCMD is a single Unicode text file, which means that UCMDs can very easily be shared. As soon as the text file defining a UCMD is copied into the UCMD folder, the user command can be called from the APL session — or invoked under program control. It is Dyalog's hope that the introduction of user commands will lead to much more widespread sharing of development tools in the APL community.

This tutorial will show how to create, debug and modify user commands and how to manage them. It will show how to deal with arguments, options and results. Several examples will be shown, from the very simple to the more complex.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

TU4: Kai Jäger (APLTeam): APLWiki

The software the APL Wiki is based on (MoinMoin) comes with an impressive number of features, but many of these features do not exactly advertise themselves. This talk provides background information regarding the APL Wiki as well as tips and tricks how to get the best out of the wiki:

- How to subscribe to changes.
- How to use the "Recent changes" tab.
- "Find" - powerful but a bit tricky.
- Why one needs a user account.
- Why email addresses on the APL Wiki are relatively save.
- The APL Wiki as a marketing tool.
- Vandalism and how this effects all of us.
- How to get help.
- Performance considerations.
- Configuration options.
- Understanding Categories.
- About APL characters.
- How to create a page.
- About passwords.
- How to create a new page.
- Formatting issues: headers, lists, tables and APL code.
- More...

Note that it is mostly about using the APL Wiki; only a small part of the information provided addresses the needs of contributors.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

TU5: Joe Blaze (APL2000): APLNext WebServices

Why Web-enable?

- Benefits to the Customer
- Benefits to the Developer/Maintainer/Vendor

Overview of web-enabling an APL+Win Application

- Re-purpose the application system into a multi-tiered architecture
- Use GUI construction technology suitable for web-clients
- Web server(s) to house the APL+Win-based application system business rules and calculations
- Application monetization options

Samples available on the APL2000 Forum

- Benefits, Licensing and Prices: http://forum.apl2000.com/viewtopic.php?t=440
- What is APL WebServices: http://forum.apl2000.com/viewtopic.php?t=358
- Beginner's Guide…: http://forum.apl2000.com/viewtopic.php?t=357
- APL WebServices Documentation: http://forum.apl2000.com/viewtopic.php?t=339
- Importing and Exporting APL WebServices Configurations: http://forum.apl2000.com/viewtopic.php?t=436
- Making the Server Visible (for development purposes) : http://forum.apl2000.com/viewtopic.php?t=435
- APLNext Application Server Installation from the msi: http://forum.apl2000.com/viewtopic.php?t=423
- APL WebServices Connection Timeout Setting: http://forum.apl2000.com/viewtopic.php?t=537
- Using a .Net WPF GUI with APL WebServices: http://forum.apl2000.com/viewtopic.php?t=360
- APL WebTransfer Class and a Proxy Server: http://forum.apl2000.com/viewtopic.php?t=368
- Using APLWebServices with an HTML or PDF client GUI: http://forum.apl2000.com/viewtopic.php?t=359
- Multi-threading, scaling, queueing, loadbalancing and state: http://forum.apl2000.com/viewtopic.php?t=356&start=0&postdays=0&postorder=asc&highlight=
- Monte Carlo Simulation Overview: http://forum.apl2000.com/viewtopic.php?t=544

\pagebreak[3] **Consider the .Net alternative**

- Use Microsoft Silverlight or most any other browser-based methodology to create the client-side GUI
- Use VisualAPL or C# to create the server-side portion of the application system
- The GUI can communicate with the server-side via WCF (Windows Communication Foundation)
- APL2000 Consulting Services can help you develop this type of application quickly and economically
- A complete example is available on the APL2000 Forum at: http://forum.apl2000.com/viewtopic.php?t=543

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Workshops

WS1: Brian Becker (Blue Dolphin Solutions): APL and Web Services

What is a Web Service?
The W3C defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards."

Web Services are modular — a Web Service is self-contained and self-describing. Everything necessary to invoke a Web Service and interpret its results is a part of the Web Service itself.

Web Services are accessed via standard protocols — Web Services can be accessed over the Internet or an intranet using a web browser or other client.

Web Services are platform independent — Web Service providers and requestors can communicate effectively without any knowledge of the platform that either is using.

Web Services share data, business logic, or processes — Web Services can deliver a wide range of function from very simple a query/response service to very complicated business processes.%

Introducing SAWS — the Stand Alone Web Service framework
SAWS is a tool developed for Dyalog APL version 12.1. SAWS makes it possible for APL programmers to integrate Web Services developed by others into their APL applications. SAWS also enables APL programmers to make the functionality of their applications available to others over the Internet. Best of all, SAWS accomplishes this without forcing the APL programmer to become an expert in the underlying protocols and technologies necessary to implement or use Web Services. SAWS can help transform your legacy APL application into a web-enabled service.

In This Workshop
We will:

- Present a brief overview of Web Services and their underlying protocols.
- Learn how to call third party Web Services to use their results in an APL application.
- Develop a simple web service using SAWS which can be invoked from SAWS, a web browser, or other client.

A non-commercial license copy of Dyalog APL 12.1 will be provided for those who need it.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

WS2: John Scholes (Dyalog Ltd.): Introduction to D-Functions

Since their introduction in 1996, Dyalog's direct-definition functions (D-fns) have grown from an experimental toy to a notation used to implement large pieces of commercial software. D-fns are not only useful for expressing idioms, but also as a tool of thought for any problem that can benefit from a functional approach (and some would say that covers almost everything). Owing to their functional nature, D-fns also have greater potential for internal optimization, including compilation — and have been selected as the foundation for the new function syntax in the APL# dialect.

The workshop will start with an easy introduction; discuss where the use of D-fns is appropriate; and finish with some fireworks.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

WS3: Michael Hughes and Morten Kromberg (Dyalog Ltd.): Windows Presentation Foundation

Windows Presentation Foundation is a graphical subsystem for rendering user interfaces, originally developed for desktop applications built using the Microsoft.Net platform - but now also available for web and mobile applications under Windows, Linux, on the Apple Macintosh - and coming to mobile platforms. The workshop will demonstrate how WPF can be used from Dyalog APL and APL# to create desktop and web applications. Participants will leave the workshop with a working WPF application that can be used as a basis for further work with WPF. The agenda consists of:

- Introduction to WPF itself - "why bother"?
- Controls/Layouts, including Methods /Properties/Attached Properties/\hspace{0ex}Events/Commands
- Different models for WPF development: Pure XAML, Code+XAML, Pure Code
- Translating xaml into code, particularly for attached properties,
- Customising xaml such as adding events (□xml)
- Finding and understanding Documentation
- Tips for translation/ data structures/ use of dyadic []class
- Property binding, Commands and Triggers
- Debugging WPF applications
- Some more interesting controls, Ribbon, DataGrid, FlowDocuments, etc
- Data binding
- Building your own controls - Inheritance
- Creating distribution dll for control and/or application
- Graphics / Animation
- Templates and Styles

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

WS4: John Daintree (Dyalog Ltd.): Using the Microsoft.Net Framework

Dyalog integrates comfortably with the Microsoft.Net Framework. This course will give an overview of, and show how you can take advantage of, the features included in the Framework itself, and in Visual Studio, Microsoft's cross-language development platform. John will show you how to find and understand documentation of the framework class libraries, and he will introduce you to some of the most useful classes. We will explore how to use the VS Form Designer to build forms which use APL code, and write APL classes which can be used from C# and VB.Net. The course will very briefly show how to call APL code from Microsoft Internet Information Services (IIS).

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Forum

FO1: Bob Smith (Chair) (Sudley Place Software): APL in 2020

While the future is notoriously hard to predict, talking about it with the leading industry experts is an excellent way to begin to picture it. As such, the idea to start an online discussion of “APL in 2020” is a highly laudable goal with the APL conference in Berlin as the next stepping stone in this sequence.

This forum expects to use the following format to cover these topics:

- An initial presentation of the why this is such an important topic, the rationale for starting the online exchange, an in-depth summary of the topics covered so far, what topics might be covered in the future, how the audience can contribute, and the next steps. This part is given by Phil Last, one of the four originators of the "APL in 2020" idea.
- A panel to discuss the previous topics, what new topics should be discussed either then or in the comp.lang.apl newsgroup, next actions, etc.
- A discussion of these topics and more with the audience. We're allotting plenty of time for audience interaction, so come prepared to discuss your ideas on this vital window into the future of APL.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

FO2: Dr. James A. Brown (Chair) (CEO, SmartArrays, Inc.): The Future of Parallel Computing with APL

For a long time, people have said that APL is just waiting for parallel hardware to become widely available. Today in 2010, parallel hardware is widely available. Four processor laptops are not expensive. The time for parallel APL is now.

This forum is for discussion of parallelism and how APL implementations can exploit parallel hardware and how vendors of APL can position their offerings to make APL a preferred choice for implementation of parallel algorithms. As part of the APLin2020 discussions, it is appropriate to pose questions and initiate topics for which conclusions cannot be achieved in one session. Discussions should continue in the APLin2020 forums and in the USENET forum comp.lang.apl.

The general topic is “What will make APL the platform of choice for parallel programming in the future? - in particular in 2020”

There will be a panel who will give opinions and respond to questions but this will also be an open session hopefully with significant contributions from the attendees.

Questions that could be addressed:

- What support do vendors of APL already provide?
- What should be added to APL to make it more acceptable?
- What should be removed from APL to make it more acceptable?
- Is automatic parallelization of algorithms enough?
- Does it matter that different vendor approach coarse grain parallel differently?
- Should APL vendors agree on common primitive operations?
- Is anything more needed by developers from the hardware platforms?
- Should function arrays be considered as a way to deliver parallelism?
- Can we realistically predict the state of computing 10 years from now?

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Selling Tutorials

ST1: Gitte Christensen (Dyalog Ltd.): APL - why, when and where

In this presentation we look at the current trends in software development and how the rest of the software world is doing. Some areas where APL is particularly competitive are identified and a list of Selling Points for APL is presented.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

ST2: Paul Grosvenor (optimasystems): Making Money with APL

To make APL or indeed any other technology pay we must be very clear and positive about what it is we are selling, where we are selling it and why our client needs it. If that wasn’t enough, and in the current climate, our customers are very sensitive to both cost and risk and these two hurdles must be addressed and overcome quickly and decisively.

In this part of the presentation we will explore a few simple techniques to bring your proposal to the top of the list. It all comes

down to confidence, preparation and rapport with a little bit of persistence and luck.

With some audience participation I aim to get you thinking 'out of the box' and demonstrate how you can make your proposal stand out from the crowd, to be so intriguing in fact, that your prospective client will be calling you.

I'll even show you how to tell someone your name in a way that stands a better chance of them remembering it.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

Vendor Sessions

APL2000

V01: Joe Blaze (APL2000): WPF Presentation & APL Business Rules Components in a Windows Application System

This session will illustrate the construction of a simple Windows Presentation Foundation (WPF) GUI component which is supported by an APL+Win or VisualAPL business rules component. Attendees with a copy of Visual Studio 2008 or 2010 and APL+Win or VisualAPL on their personal computers should be able to "follow along" with the presentation and construct their own copy of the project.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V02: Patrick Parks (APL2000): APL+Win V10 Enhancements in Detail

This session will expose additional significant enhancements implemented in APL+Win V10 including:

- Workspace memory improvements and new performance tuning features
- Maximum array size improvements for larger arrays
- Separate debug and release modes for better application system quality control
- Enhanced error handling and recovery, trace log and testing features
- New control structures for improved source code clarity
- Unicode clipboard support

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V03: Patrick Parks (APL2000): APL+Win V10 Interpreter Performance Enhancement in Detail

This session will expose the interpreter performance improvements in APL+Win V10 including their implementation strategy and some measurements of the results.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V04: Joe Blaze (APL2000): APLNext Supervisor — A Simple Example

This session will expose an easy-to-understand example using the APLNext Supervisor to employ instances of APL+Win as ActiveX servers in a multi-threaded manner.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V05: Joe Blaze (APL2000): APL2000 Customer Forum

APL2000 customers are cordially invited to attend this session to present their suggestions and questions about APL2000 products. The APL2000 staff attending the APL 2010 Berlin conference will be present to engage in the discussion including the chief system architect for the APL+Win product, Patrick Parks, the director of sales & marketing, Sonia Beekman, and the managing director, Joe Blaze.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

V06: Joe Blaze (APL2000): APLNext VisualAPL — Programming Examples

This session will expose a selection of VisualAPL programming examples including:

- Using the Cielo Explorer interactive session to explore .Net components
- Referencing and calling a VisualAPL .Net assembly from C#
- Multi-threading in VisualAPL
- Runtime use of a Cielo Explorer VisualAPL script in C#
- Using the APLNext Lightweight Array Engine (LAE) in CE
- XML Serialization in VisualAPL
- Using VisualAPL to Automate Microsoft Excel
- Converting APL+Win source code to VisualAPL
- Exposing VisualAPL functions as ActiveX methods
- Using VisualAPL from WPF
- Using VisualAPL from APL+Win
- Using APL+Win from VisualAPL

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

V07: Joe Blaze (APL2000): APLNext WebServices — A Practical Example

This session will expose a production application system using a substantial WPF GUI component which is supported by an APL+Win business rules component exposed to the presentation layer via APLNext WebServices. The GUI component collects a significant amount of technical data in xml-format which is parsed and processed by the APL+Win business rules component into pdf-format documents that are returned to the presentation component via APLNext WebServices.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

V08: Joe Blaze (APL2000): “MVC” and “Presentation Model” System Architecture for APL

This session will expose certain application system architecture styles currently being adopted by mainstream programmers with an emphasis on their design rationale and how they may affect APL-based programming.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

Dittrich & Partner Consulting

V09: Klaus-Peter Friedrich (Rheinischer Sparkassen- und Giroverband (RSGV)): STS.win - An APL2 OLAP Database

STS.win is a powerful self-developed application of RSGV for managing and analyzing statistical information. In this presentation a short introduction is given for the motivation and concept behind. Main part will be a live demonstration of STS.win in action.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

V10: Katrin Holzmüller and Vladimir Zakgeym (DPC GmbH): APL2 at a Young Glance

This presentation focuses on the first impressions of language and programming environment of APL-newcomers with different professional backgrounds. Stumbling blocks and light bulb moments of the first programming attempts are described as well as benefits and troubles. Finally a few ideas are outlined how to facilitate the introduction of newcomers to APL.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

Dyalog Ltd.

V11: John Daintree (Dyalog Ltd.): Taking APL for a RIDE

The Dyalog Remote Integrated Development Environment (RIDE) is a new graphical development environment for all versions of Dyalog APL on all platforms - from Windows Mobile to the largest AIX “midframes” (and in the future also the new APL# interpreter). The RIDE allows you to connect to Dyalog session from almost any web browser on any client platform. The RIDE is a cornerstone of a strategy which is intended to take the portability of applications written in Dyalog APL “to the next level”.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V12: Morten Kromberg (Dyalog Ltd.): Dyalog Technical Keynote

Morten will review Dyalog APL Version 13.0, which is about to enter “Beta Test”, and talk about APL#, the RIDE, and the rest of Dyalog's "Road Map" for 2011 and beyond.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V13: Jay Foad (Dyalog Ltd.): An interpreter for Vanilla Siteswap

Jay is the newest arrival at Dyalog. He secured the job by juggling during the job interview, while talking us through C code which interpreted a "Domain Specific Notation" for juggling known as Vanilla Siteswap (with synchronous + multiplexing extensions). Jay will be presenting his new VS/APL interpreter.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V14: Morten Kromberg (Dyalog Ltd.): Your Application as an SQL Data Source

Dyalog has developed a prototype of a framework which allows your application to look, taste and smell like a relational database. The SQAPL Server allows reporting on live APL data and analytical output from tools like Microsoft Excel or - Access, Crystal Reports - and virtually any programming language - in fact, any client application which can use ODBC, JDBC, ADO or ADO.NET. The interface fully supports SQL, including database updates and inserts, your APL functions as “stored procedures”, and creation/modification of tables (DDL statements) - and requires very little SQL expertise to set up.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V15: Kai Jäger (APLTeam): APL2XML

HelpAndManual (http://www.helpandmanual.com/) is by far the best software available for creating any sort of documentation. The software comes with a powerful editor that allows the user comfortably to edit all sorts of aspects of documentation, including references, a glossary, an index and a table of contents as a tree-structure. The application saves it's data in XML files. The application compiles these files not only into ordinary Windows help files but also compiled help, RTF documents, PDF documents, web sites and electronic books.

That fact that it saves its internal data in ordinary XML files means that any application could create such XML files itself and then use HelpAndManual as a compiler in order to generate the preferred output format.

This seems to be particularly useful for classes: plenty of useful information can be generated from class scripts automatically: a list of methods including their syntax as well as a list of properties and fields.

In the case where a class script contains information about itself following some simple and basic rules, these pieces of information can be used as well.

This talk demonstrates how this can be achieved by using a couple of classes which are already available as part of the APLAPL project on the APL Wiki (ADOC; see http://aplwiki.com/ADOC) or will become available shortly before the APL2010 conference: the APL2XML classes.

With these classes one can create XML files which will be accepted by HelpAndManual as input files. Before the APL2010 conference ADOC will be enhanced to make use of the APL2XML classes in order to create any of the output formats supported by HelpAndManual.

Apart from its usefulness the approach demonstrates clearly the advantage of using XML as file format. This allows applications to work together although they don't know much about each other.

The advantages of an object-oriented approach in software development also becomes apparent: Both ADOC as well as the APL2XML classes are quite complex, but the complexity is hidden: the user is expected to deal only with the public interface which is easy and clean.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V16: Stig Nielsen (SimCorp A/S): Migrating SimCorp Dimension to Dyalog APL Unicode

SimCorp Dimension is an integrated single database investment management system mainly coded in Dyalog APL, partly in C#. Due to market demands, the APL and database part of the system has been migrated to support Unicode. Stig will take you through the steps needed, and the pitfalls to be aware of, to get successful through the migration process and finally release the product. The main challenge was to get all the interfaces to e.g. external C-libraries, third party products and general native file access issues. A general introduction to what Unicode is and why it is, or is not, the answer to all problems with multilingual challenges when it comes to sharing data will start the session.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V17: Ryan Tarpine and Mstislav Elagin: Winning the Dyalog Programming Contest 2010

Ryan is the winner of the Dyalog Programming Contest 2010. He is a 25-year-old PhD Candidate in Computer Science with a focus on computational biology from Brown University near Boston. The runner-up in the contest is Mstislav, a 32-year old PhD student at the Humboldt-Universität zu Berlin, who is currently developing an online monitoring and early warning system for financial markets. Ryan and Mstislav will speak about their experiences in learning APL - and using it to win prizes!

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

IBM APL Products and Services

V18: David Liebtag (IBM APL Products and Services): Using APL2 with Java and the WebSphere Application Server

The Java Platform, Enterprise Edition (JEE) is a widely used platform for server applications. In particular JEE servers such as the Apache server and the WebSphere Application Server maintain a significant share of the server market. Associated Processor 14, APL2's interface for calling Java, and the Calls to APL2 Java interface enable APL2 applications to fit seamlessly in JEE server applications. This presentation will introduce these facilities and demonstrate using APL2 in the WebSphere Application Server environment.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---

V19: David Liebtag (IBM APL Products and Services): Recent APL2 Enhancements

This presentation will introduce and demonstrate two recent APL2 enhancements, the Parallel Each operators which run functions on multiple processors and machines, and Associated Processor 15 which supports strong data typing and monitoring changes to variables.

Goto top of page top of table of contents schedule Monday Tuesday Wednesday Thursday

---