2008 Lang.NET Symposium

Jason Zander

General Manager, Visual Studio

Jason Zander

General Manager, Visual Studio

Visual Studio Pro & Standard

C#

Core IDE and VSIP

Visual Basic

Popfly & Non-pro

Visual C++

Mobile Tools & .NET CF

Phoenix

Javascript

IronPython, IronRuby, DLR

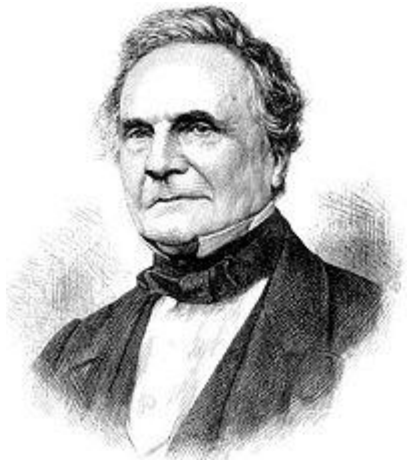
Office Developer Tools

Incubations…

Welcome!!

- Wide industry representation
- Provide insight on our plans
- 50% non-Microsoft content
- Sharing ideas, debating approaches

A Little History


Charles Babbage

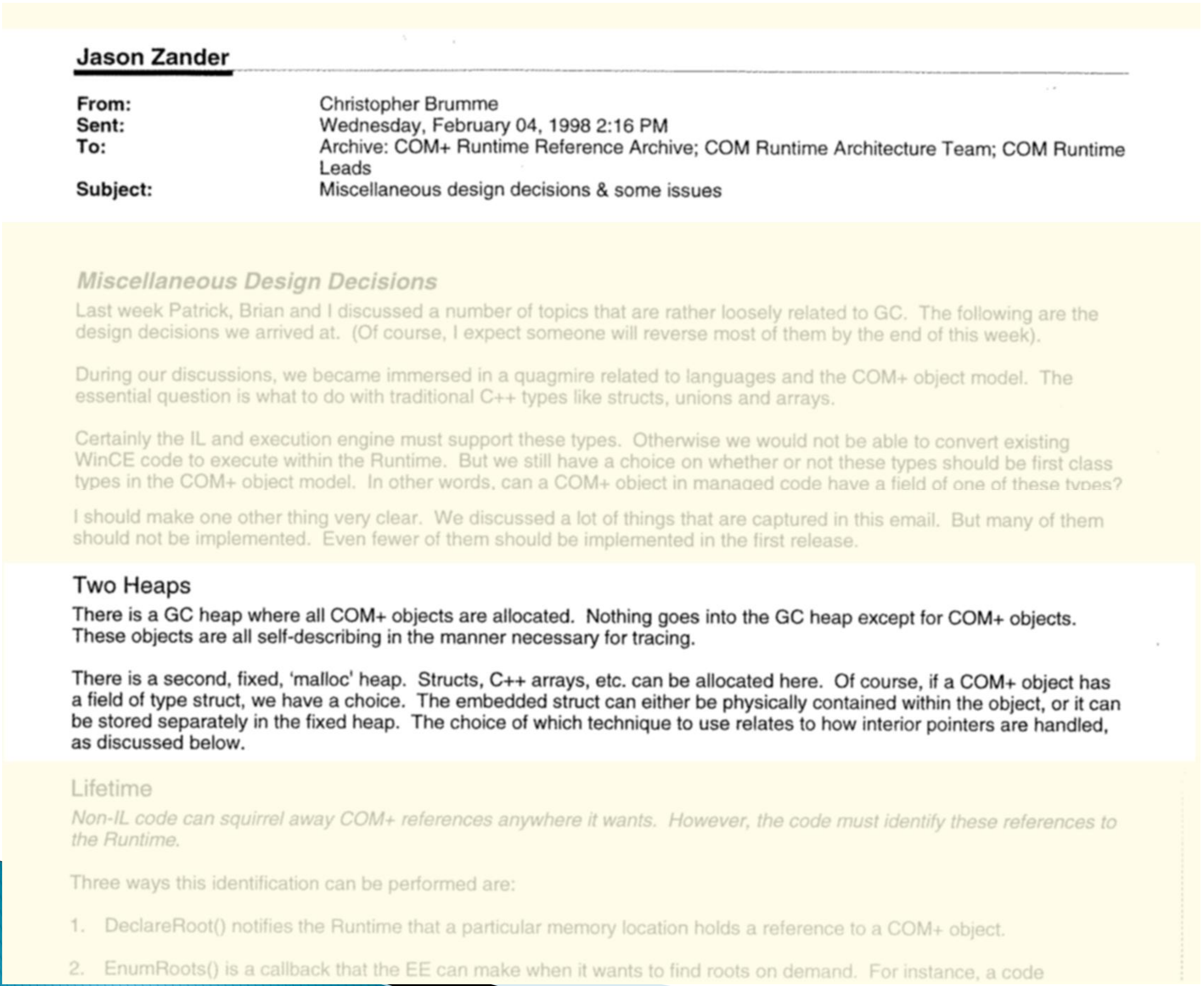
Ada Lovelace

- Thousands of languages
- Lot’s of great ideas to borrow from
- Hopefully most of them good…

  Algol 60 was not only an improvement on its predecessors, but also on nearly all its successors. (C. Hoare)

Some Goals for the CLR / .NET

- Modernize Microsoft programming interfaces
- Create a consistent programming interface across all form factors
  - Device, desktop, server, database, browser…
- Stop building tons of forms packages (MFC, Ruby (VB), etc)
- Allow developers to utilize their skills orthogonal to language choice
  - Provide a common platform for all languages
  - CLI/CLS
- Improve productivity



Jason Zander

From: Christopher Brumme
Sent: Wednesday, February 04, 1998 2:16 PM
To: Archive: COM+ Runtime Reference Archive; COM Runtime Architecture Team; COM Runtime Leads
Subject: Miscellaneous design decisions & some issues

Miscellaneous Design Decisions

Last week Patrick, Brian and I discussed a number of topics that are rather loosely related to GC. The following are the design decisions we arrived at. (Of course, I expect someone will reverse most of them by the end of this week).

During our discussions, we became immersed in a quagmire related to languages and the COM+ object model. The essential question is what to do with traditional C++ types like structs, unions and arrays.

Certainly the IL and execution engine must support these types. Otherwise we would not be able to convert existing WinCE code to execute within the Runtime. But we still have a choice on whether or not these types should be first class types in the COM+ object model. In other words, can a COM+ object in managed code have a field of one of these types?

I should make one other thing very clear. We discussed a lot of things that are captured in this email. But many of them should not be implemented. Even fewer of them should be implemented in the first release.

Two Heaps

There is a GC heap where all COM+ objects are allocated. Nothing goes into the GC heap except for COM+ objects. These objects are all self-describing in the manner necessary for tracing.

There is a second, fixed, 'malloc' heap. Structs, C++ arrays, etc. can be allocated here. Of course, if a COM+ object has a field of type struct, we have a choice. The embedded struct can either be physically contained within the object, or it can be stored separately in the fixed heap. The choice of which technique to use relates to how interior pointers are handled, as discussed below.

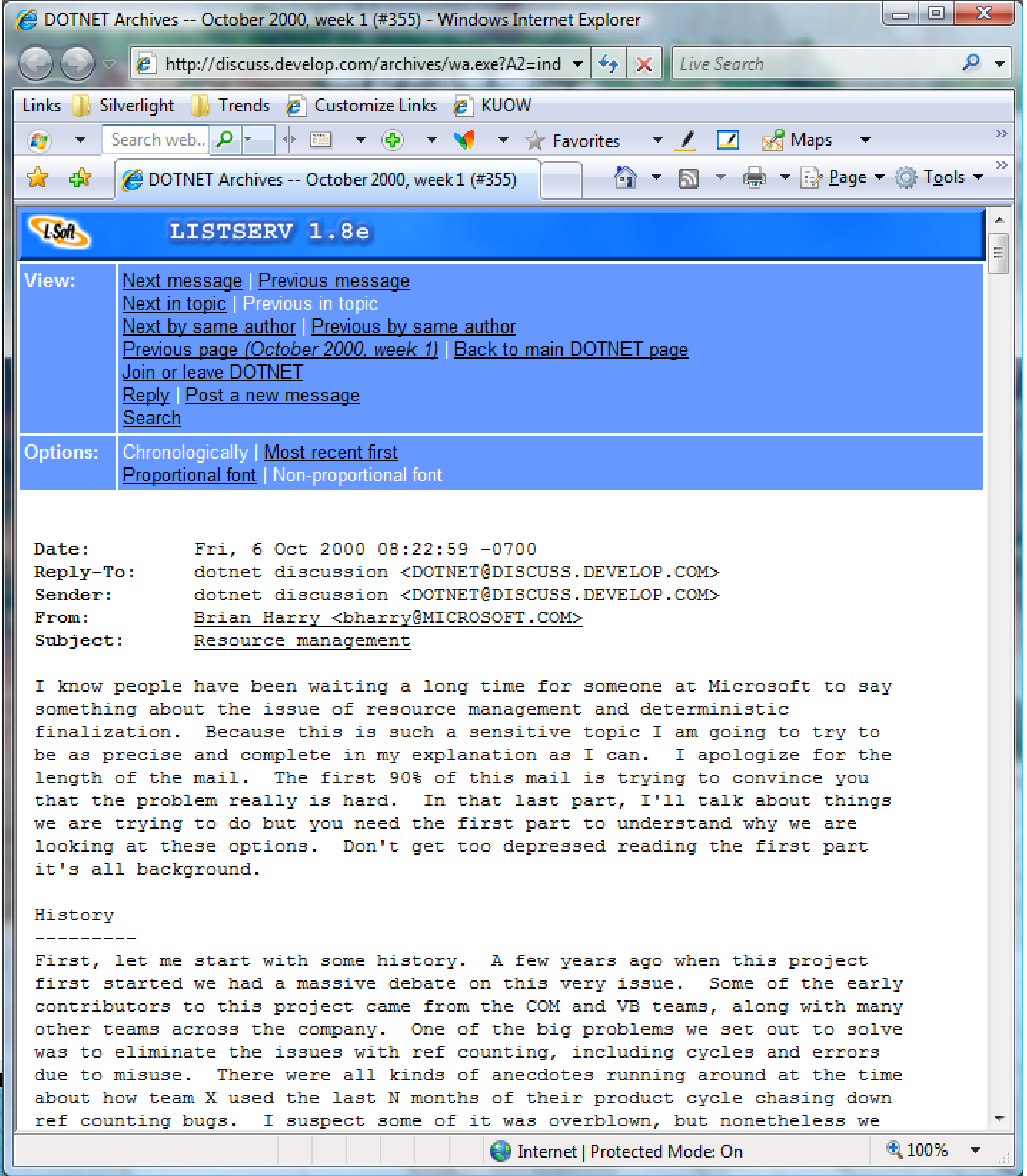
Lifetime

Non-IL code can squirrel away COM+ references anywhere it wants. However, the code must identify these references to the Runtime.

Three ways this identification can be performed are:

1. DeclareRoot() notifies the Runtime that a particular memory location holds a reference to a COM+ object.

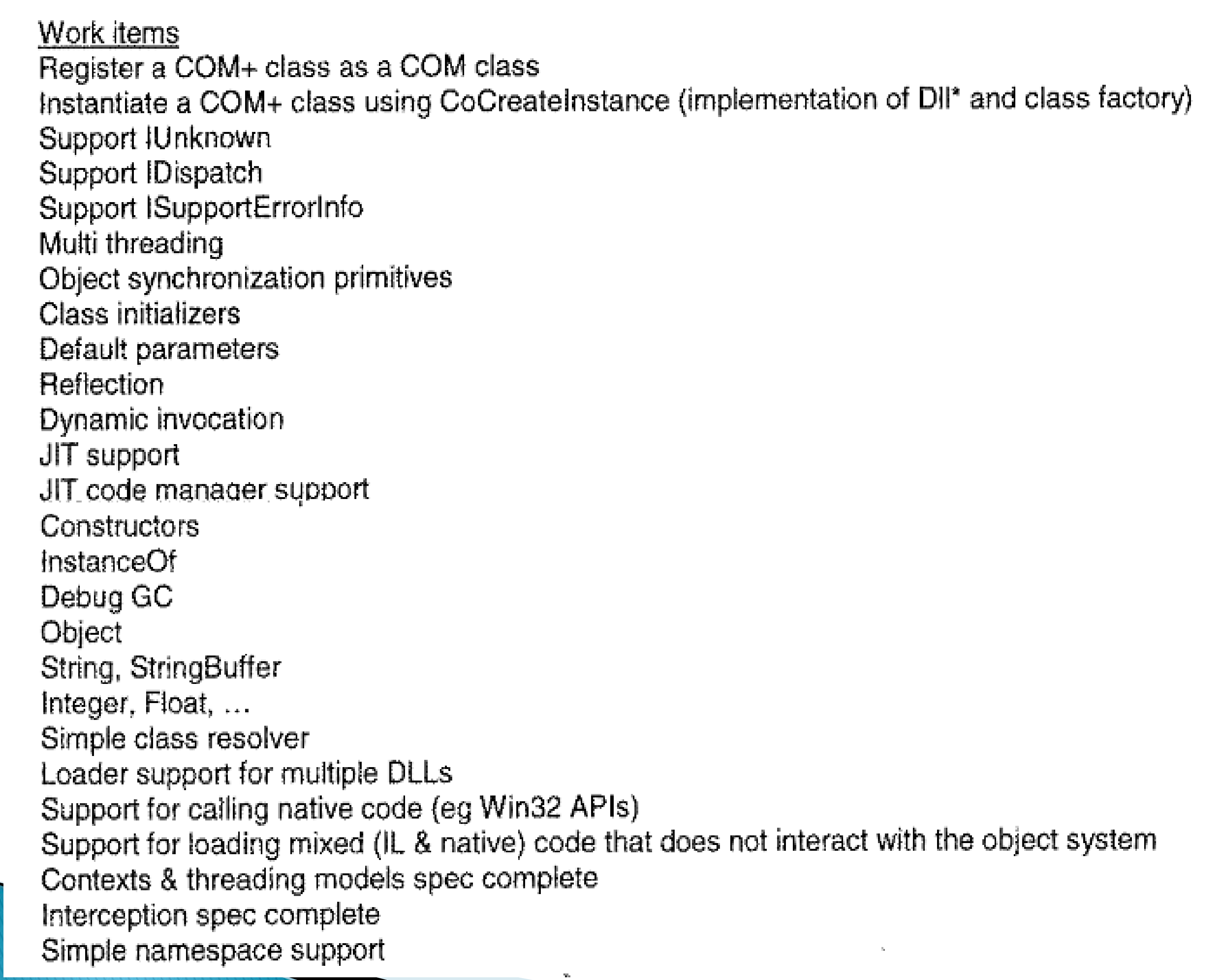
2. EnumRoots() is a callback that the EE can make when it wants to find roots on demand. For instance, a code


DOTNET Archives -- October 2000, week 1 (#355) - Windows Internet Explorer
http://discuss.develop.com/archives/wa.exe?A2=ind
Live Search
Links Silverlight Trends Customize Links KUOW
Search web..
Favorites
Maps
DOTNET Archives -- October 2000, week 1 (#355)
Page Tools
LISTSERV 1.8e
View:
Next message | Previous message
Next in topic | Previous in topic
Next by same author | Previous by same author
Previous page (October 2000, week 1) | Back to main DOTNET page
Join or leave DOTNET
Reply | Post a new message
Search
Options:
Chronologically | Most recent first
Proportional font | Non-proportional font
Internet | Protected Mode: On
100%

```
Date:          Fri, 6 Oct 2000 08:22:59 -0700
Reply-To:      dotnet discussion <DOTNET@DISCUSS.DEVELOP.COM>
Sender:        dotnet discussion <DOTNET@DISCUSS.DEVELOP.COM>
From:          Brian Harry <bharry@MICROSOFT.COM>
Subject:       Resource management

I know people have been waiting a long time for someone at Microsoft to say
something about the issue of resource management and deterministic
finalization.  Because this is such a sensitive topic I am going to try to
be as precise and complete in my explanation as I can.  I apologize for the
length of the mail.  The first 90% of this mail is trying to convince you
that the problem really is hard.  In that last part, I'll talk about things
we are trying to do but you need the first part to understand why we are
looking at these options.  Don't get too depressed reading the first part
it's all background.

History
---------
First, let me start with some history.  A few years ago when this project
first started we had a massive debate on this very issue.  Some of the early
contributors to this project came from the COM and VB teams, along with many
other teams across the company.  One of the big problems we set out to solve
was to eliminate the issues with ref counting, including cycles and errors
due to misuse.  There were all kinds of anecdotes running around at the time
about how team X used the last N months of their product cycle chasing down
ref counting bugs.  I suspect some of it was overblown, but nonetheless we
```


Work items
Register a COM+ class as a COM class
Instantiate a COM+ class using CoCreateInstance (implementation of Dll* and class factory)
Support IUnknown
Support IDispatch
Support ISupportErrorInfo
Multi threading
Object synchronization primitives
Class initializers
Default parameters
Reflection
Dynamic invocation
JIT support
JIT code manager support
Constructors
InstanceOf
Debug GC
Object
String, StringBuffer
Integer, Float, …
Simple class resolver
Loader support for multiple DLLs
Support for calling native code (eg Win32 APIs)
Support for loading mixed (IL & native) code that does not interact with the object system
Contexts & threading models spec complete
Interception spec complete
Simple namespace support

Key Feature Progression

- V1 was very powerful
  - Full type system
  - Compilation functionality
  - Full class libraries
  - Full tools support
  - Support for scripting languages
  - ASP.NET for web, Windows Forms for client
- V2 added
  - Additional robustness, allowing in-process SQL Database support
  - Edit & Continue
  - Full generics support
- Future improvements built on prior investments
  - E.g. LINQ building on generics
- Several form factors
  - .NET Compact Framework for devices & Xbox 360
  - Silverlight for browser

Dynamic Languages

- Procedural, Functional, and Dynamic languages were a goal from day one
  - Example: tail prefix for call instructions
- Additional functionality added in V2
  - Performance work
  - Lightweight Code Generation (LCG)
- Dynamic Language Runtime (DLR) allows for even easier integration in the future

Sharing Source Code

- Long history with SSCLI, aka “Rotor”
  - Shared source, academic friendly license
- Ultimately led to release of .NET Framework source under MS-RL
- IronPython OSS compatible license took months
- Microsoft Permissive License (MS-PL) made it easier for future releases
  - Shipped ASP.NET AJAX Control Toolkit under
  - IronRuby
  - Dynamic Language Runtime (DLR)
- Now taking contributions for IronRuby libraries

Trends

- Making web programming easier
  - Go for reach or rich environment?
- Integration of query logic
- Easing integration of markup and imperative logic
- Parallel computing advances

Have a great conference!