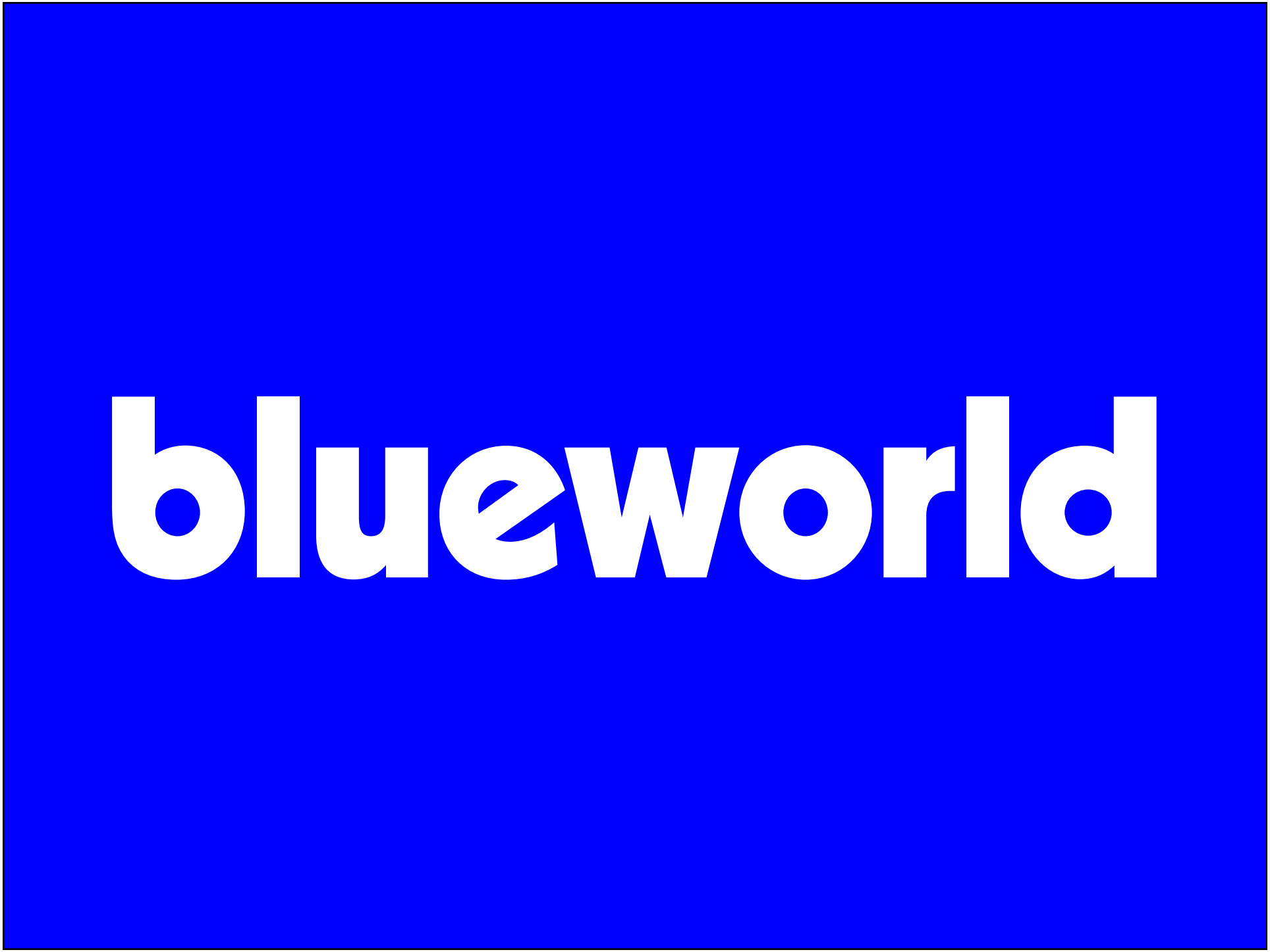

blueworld

Advanced Developer Techniques

blueworld

Introduction
Speakers

- **Fletcher Sandbeck**
  Lasso Product Specialist

- **Kyle Jessup**
  Lead Engineer

blueworld

Introduction
Topics

- Compound Expressions
- Logging Controls
- Global Variables and References
- Map Tags
- Symbol Overloading
- Unknown Tags

- Lasso Developer 6 Technology Preview

blueworld

Introduction
Goals

- Demonstrate New Features of Lasso Professional 6.
- Code maintenance and debugging techniques.
- Reduce memory use and execution time.
- Create professional quality tags and data types.

- Demonstrate new features of Lasso Developer 6.

blueworld

Compound Expressions
Goals

- **Allow complex expressions to be embedded within tag calls.**

  [Var: 'result' = [If: $condition]Alpha[Else]Beta[/If]]

- **Allow developers to easily reuse code.**
- **Make tags a first class data type so they can be passed as parameters.**

  [exArray: $array, [tag]]

- **Easy execution of asynchronous processes.**

blueworld

Compound Expressions
Characteristics

- **Delimited by curly brackets { }.**

  { Compound Expression }

- **LassoScript syntax, tags separated by ;.**

  { (Tag: parameters); (Tag: parameters); }

- **Specify return value using [Return] tag.**

  { Return: 'Value'; }

- **Evaluate compound expressions using [{}->Eval].**

  [{ Compound Expression }->Eval]

Compound Expressions
Embedded Expressions

- **Goal – Allow complex expressions to be embedded within tag calls.**
- **Example – Using an [If] … [Else] … [/If] expression as a parameter.**

```
[Var: 'test' = { If: ($condition == Null);
    Return: '';
  Else;
    Return: $condition;
  /If; }->Eval]
```

blueworld

Compound Expressions
Examples

```
[Field: { If: ($fieldname == Null); Return: 'ID';
  Else; Return: $fieldname ; /If; }->Eval]
```

```
[Var: 'myArray' =  { If: ($variable->type != 'array');
  Return: (Array: $variable); Else;
  Return: $variable; /If; }->Eval]
```

```
[Var: 'found_count' = { Inline: -FindAll,
  -Database='Contacts', -Table='People' ;
    Return: (Found_Count);
  /Inline; } ]
```

blueworld

Compound Expressions
Reusable Expressions

- **Goal – Allow developers to easily reuse code.**
- **Custom Tags allow code to be reused:**

```
[Define_Tag: 'myTest', -Required='condition']
  [If: #condtion == Null]
    [Return: '']
  [Else]
    [Return: #condition]
  [/If]
[/Define_Tag]

[Var: 'result' = (myTest: $condition)]
[Var: 'result2' = (myTest: $condition2)]
```

blueworld

Compound Expressions
Reusable Code

- **Compound expressions can be stored in a variable and run as a tag using [{}->Run].**

```
[Var: 'myTest' = { If: #condtion == Null; Return: '';
  Else; Return: #condition; /If; } ]

[Var: 'result' = $myTest->(Run: -Params=$condition)]
[Var: 'result2'=$myTest->(Run: -Params=$condition2)]
```

Pro – This requires less code than creating a tag for small code snippets.

Con – Requires use of the [{}->Run] tag.

blueworld

Compound Expressions
Reusable Code

- **Goal – Make tags a first class data type.**

  Allow tags to be created, renamed, and replaced programmatically.
- **[Tags] is a map of tag names and code.**
- **Compound expressions can be installed into the [Tags] map and then called as custom tags.**

blueworld

Compound Expressions
Reusable Code

- **Compound expressions can be stored in the [Tags] map and then called as a custom tag.**

```
[Var: 'myTest' = { If: #condtion == Null; Return: '';
  Else; Return: #condition; /If; } ]
[Tags->(Insert: 'myTest' = $myTest)]

[Var: 'result' = (myTest: $condition)]
[Var: 'result2'=(myTest: $condition2)]
```

A custom tag created in this way is no different than a custom tag created using the [Define_Tag] … [/Define_Tag] tags.

Compound Expressions
Expression Parameters

- **Goal – Make tags a first class data type so they can be passed as parameters.**
- **Compound expressions can be stored in variables.**
- **Compound expressions can be passed as parameters.**
- **Create a custom tag which accepts a compound expression as a parameter and applies that expression to each element of an array.**

```
[exArray: (Array: 1, 2, 3, 4, 5),
  { Return: (Params)->(Get:1) • 2; }]
```

➜ (Array: 2, 4, 6, 8, 10)

blueworld

Compound Expressions
Expression Parameters

- **The [exArray] tag takes two required parameters: an array and a compound expression.**
- **The expressions is applied to each element of the array in turn.**
- **The resulting value is stored back in the array.**

```
[Define_Tag: 'exArray', -Required='array',
    -Required='expression']
  [Iterate: #array, (Local: 'item')]
    [#item = #expression->(Run: -Params=#item)]
  [/Iterate]
[/Define_Tag]
```

blueworld

Compound Expressions
Asynchronous Operations

- **Goal – Easy execution of asynchronous code.**
- **This is a replacement for the [Post_Inline] functionality from Lasso WDE 3.x, but is more reliable and flexible.**
- **Use [{ }->AsAsync] to execut code asynchronously.**
- **The [Sleep] tag can be used to delay execution a number of milliseconds.**

```
<?LassoScript
  { Sleep: 1000;
    Log_Critical: 'Executing one second later'; }->AsAsync;
?>
```

blueworld

Logging Controls
Introduction

- Lasso Professional 6 introduces three log levels and destinations.
- Error messages can be flagged according to their importance.
- The global administrator can decide what error messages they want to see in each of three destinations.
- More information can be viewed while debugging. Then, only important errors can be viewed once a site goes live.

blueworld

Logging Controls
Log Levels

- **Critical – Issues which must be brought to the attention of the global administrator. These issues affect the proper operation of Lasso Service.**
- **Warning – Issues that provide information about the state of Lasso Service or which may point to an error on a particular page, but not a system-wide error.**
- **Detail – Detailed messages about the proper operation of a server. Debugging messages.**

blueworld

Logging Controls
Example

- All SQL statements issued are logged as Detail messages.
- Invalid SQL statements are logged as Warning messages.
- If Lasso MySQL is not available then an Error message is logged.

blueworld

Logging Controls
Log Destinations

- **File – The LassoErrors.txt file in the same folder as Lasso Service.**

  tail -f LassoErrors.txt

- **Console – The Lasso Service console.**

  consoleLassoService.command
  LassoService.exe

- **Database – View in the monitor section of Lasso Administration.**

blueworld

Logging Controls
Setting Destinations

- Change the destination using the monitor section of Lasso Administration or using [Log_SetDestination].
- Routing can be changed on the fly depending on what information you need immediately.
- All messages can be routed to database for remote debugging.
- LassoErrors.txt can be watched without restarting Lasso Service to view the console.

blueworld

Logging Controls
Log Tags

- **Log different error levels using:**

  [Log_Critical: 'Critical Error Message']

  [Log_Warning: 'Warning Message']

  [Log_Detail: 'Detail Message']

- **Note difference from traditional log tags.**

  [Log: -Window] Message for Console [/Log]

- **The advantage of the new tags is they can be easily used in custom tags and LassoScripts.**

- **The old [Log] … [/Log] tags are still required to log to files.**

Logging Controls
Summary

- New log tags in Lasso Professional 6 can be used to set the level of error messages.
- Error messages at each level can be routed to three destinations: database, file, or Lasso Service console.
- Professional solutions should make use of all three error levels to provide the best end-user experience possible.

blueworld

Global Variables & References

Goals

- Allow data to be shared between pages.
- Prevent duplication of data on a page.
- Allow easier manipulation of complex data types.
- Improve performance.
- Decrease memory footprint.
- Allow developers to create professional solutions.

blueworld

Global Variables
Introduction

- **Lasso Professional 6 has new tags which makes creating server-wide global variables easy.**
- **Create a new global:**

  [Global: 'myGlobal'='my value']
- **Access the global (on same page or any other):**

  [Global: ‘myGlobal’]
- **Check whether a global has been defined:**

  [Global_Defined: 'myGlobal']
- **Map of all globals:**

  [Globals]

blueworld

Global Variables
Examples

- **Storing server-wide preferences.**
- **Create your own session system.**
- **Control access to shared resources (Thread Tools).**

  [Semaphore] [RWLock] [Lock]
- **Sending data between threads.**

  [Event] [Pipe]
- **Caching page contents.**

blueworld

Global Variables
Notes

- **Any variables created within Lasso Startup are global variables.**
- **Use the following to 'hide' variables within Lasso Startup.**

  { Local: 'myLocal'='my value'; }->Run;

- **[Global] and [Global_Defined] are custom tags whose source code can be found in Startup.LassoApp.**

blueworld

References
Example

- **Create variables Alpha and Beta.**

  [Var: ‘Alpha’ = ‘my value’]
  [Var: ‘Beta’ = $Alpha]

- **Print out the values of Alpha and Beta**

  Alpha - my value
  Beta - my value

- **Change the value of Alpha**

  [Var: ‘Alpha’ = ‘my new value’]

- **Print out the values of Alpha and Beta**

  Alpha - my new value
  Beta - my value

References
Example

- **Create variable Alpha and set Beta to a reference.**

  [Var: ‘Alpha’ = ‘my value’]
  [Var: ‘Beta’ = (Reference: $Alpha)]

- **Print out the values of Alpha and Beta.**

  Alpha - my value
  Beta - my value

- **Change the value of Alpha.**

  [Var: ‘Alpha’ = ‘my new value’]

- **Print out the values of Alpha and Beta.**

  Alpha - my new value
  Beta - my new value

blueworld

References
Notes

- References allow two variables to point at the same underlying data object.
- Developers can decide what data needs to be copied and what data can be shared.
- Particularly useful when working with arrays and maps since a reference to either an entire complex data object or just one element can be created.
- Often used in custom tags and with global shared values.

References
Example

- **The [Iterate] ... [/Iterate] tag in LP6 sets a variable to the value of each element of a complex data type in turn.**

```
[Var: 'myArray' = (Array: 'Red', 'Blue', 'Green')]
[Iterate: $myArray: (Var: 'item')]
  [$item->(Lowercase)]
[/Iterate]
[Output: $myArray]
```

➜ red, blue, green

- **Note – The array elements are not duplicated. Each is modified in place. This takes about half the memory of creating the elements in a new array.**

Caching Example
Introduction

- **This is an example of using references and server-wide global variables.**
- **Source code is in the conference manual or on the CD.**
- **Goal – To cache a portion of a page and only re-fresh it periodically.**

  [exCache: -Name='myCache', -Expires=3600]
  ... contents ...
  [/exCache]

- **The contents of the tags will be cached. Each time the page is loaded the time stamp of the conents is checked and either the cached content is served or new contents is created.**

blueworld

Caching Example
Road Map

The code for the tag is split into four parts:

1. **Create a custom container tags**
2. **Creating the server-wide global variable to store the cache**
3. **Fetch the named item from the global cache and check the expiration date/time**
4. **Serving the cached data or generating new data**

Caching Example
Step 1

Create a custom container tag.

```
[Define_Tag: 'exCache', -Container, -Required='Name',
    -Optional='Expires']
  ...
[/Define_Tag]
```

- **[Local:'Name'] contains the value of the -Name parameter automatically.**
- **[Local:'Expires'] contains the value of the optional -Expires parameter if it is specified.**
- **[Run_Children] will return the processed value of the contents of the container tags.**

blueworld

Caching Example
Step 2

Create the server-wide global variable using the [Global_Defined] and [Global] tags.

```
[If: !(Global_Defined: 'Ex_Cache_Storage')]
  [Global: 'Ex_Cache_Storage' = (Map)]
[/If]
```

- **Store a local reference to the global variable for our convenience.**

```
[Local: 'storage' = @(Global: 'Ex_Cache_Storage')]
```

- **Now we can reference our global variable as #storage rather than as (Global:'Ex_Cache_Storage').**

blueworld

Caching Example
Step 3

Fetch the named item from the global cache.

```
[Local: 'cache' = @(#storage->(Find: #name))]
```

- **This references an element of the global variable by reference. Any changes made to the reference will be reflected in the global variable automatically.**
- **Date tags are used to check whether the stored date of the last cached content is greater than the expiration time in seconds.**

blueworld

Caching Example
Step 4

Serving new or cached data

- **New data is served by calling [Run_Children] to execute the contents of the container tags. The generated data is stored in the cache using**

  [#cache->(insert: ‘contents’ = (run_children))]

- **Note that this goes all the way into the global cache through the references.**
- **Cached data is served by returning the value of the following:**

  [#cache->(find: ‘contents’)]

blueworld

Caching Example
Examples

- **Fetch Apple's stock price once an hour.**

```
[exCache: -Name='stockdisplay', -Expires=3600]
  Apple's stock price: [Stock_Quote: 'AAPL'] as of [Date].
[/exCache]
```

- **Refresh a list once a day from a database.**

```
[exCache: -Name='soupsoftheday', -Expires=(3600 • 24)]
  [Inline: -Search, -Database='Soups', -Table='Schedule',
      'Soup_Date'=Date->(Format: %Q)]
    [Records]
      <br>[Loop_Count]: [Field: 'Soup_Name']
    [/Records]
  [/Inline]
[/exCache]
```

blueworld

Global Variables & References

Summary

- Global variables are a tool for sharing data between Web pages on the same server.
- References are a tool for sharing data between variables on a single page.
- Both can be used by developers to decrease execution time and memory usage.
- Globals can be set and retrieved just like variables using the [Global] tags in LP6.
- References are automatically used by the [Iterate] ... [/Iterate] tags.

blueworld

Map Tags
Useful Techniques

- **Two new map tags in Lasso Professional 6 are very useful for debugging maps.**
- **The following tag returns a list of all the keys stored in a map:**

  [Map->Keys]

- **The following tag returns an array of all the value stored in a map:**

  [Map->Values]

blueworld

Map Tags
Useful Techniques

- **These tags can be used on the [Locals], [Variables], or [Globals] maps to provide a list of all defined variables.**
- **List all global variables using:**

  [Globals->Keys]
- **List all page variables using:**

  [Variables->Keys]
- **List all local variables within a custom tag using:**

  [Locals->Keys]

blueworld

Data Types
Symbol Overloading & Unknown Tags

- Lasso Professional 6 provides new tools for creating professional quality data types.
- Symbol overloading allows the + - • / % symbols to be customized.
- Unknown tag processing allows tags with any name to be processed for a particular data type.
- Data types created using these tools can have the full functionality of built-in data types.
- Source code for these examples are in the summit manual or on the CD.

blueworld

Symbol Overloading
Example

- **Create a mathematical vector data type that stores ordered sequences of numbers.**

  [Var: ‘myVector’ = (exVector: 1, 2, 3, 4, 5)]

- **Allow the built-in math symbols to manipulate vectors.**

  [Output: $myVector • 2]

➜ (exVector: 2, 4, 6, 8, 10)

Symbol Overloading
Road Map

- **Define a custom tag within the definition of the data type with the name +.**

  [Define_Tag: '+'] … [/Define_Tag]

- **This custom tag is called when a + expression is evaluated with an [exVector] on the left.**

  [Output: (exVector: ...) + 4]

- **Need to handle addition of another [exVector] or addition of a literal value.**

Symbol Overloading
+ Symbol

```
[Define_Tag: '+', -Required='item']
  [Local: 'result' = (exVector)]
  [If: (#item->type == self->type)]
    [loop: self->size]
      [#result->(insert: self->(get: loop_count) +
        #item->(get: loop_count))]
  [/Loop]
  [Else]
    [Loop: self->size]
      [#result->(insert: self->(get: loop_count) + #item]
    [/Loop]
  [/If]
  [Return: #result]
[/define_tag]
```

blueworld

Symbol Overloading
Possibilities

- The + - • / % >> and += -= •= %= symbols can all be overloaded.
- The == != < > symbols can all be overloaded using the onCompare callback tag.
- The = symbol can be overloaded using the onAssign callback tag.
- Consult the Extending Lasso 6 Guide for more information.

blueworld

Unknown Tags
Example

- **Create a tag that returns the hexadecimal value for any standard HTML color.**

  [exColor->Red] -> #ff0000
  [exColor->AntiqueWhite] -> #FAEBD7

- **This data type will have well over a hundred custom tags and it is laborious to create them all.**
- **Use the _UnknownTag callback tag to define all the custom tags at once.**

Unknown Tags
Road Map

- **Define a custom tag within the definition of the data type with the name _unknowntag.**
- **The [Tag_Name] tag allows the name of the tag which is being executed to be retrieved.**
- **Within the custom tag find the value for a color from a map of all the standard HTML colors.**

```
[Global: 'excolor_colors' = (Map: 'white'='#ffffff',
  'red'='#ff0000', 'green'='#00ff00', 'blue'='#0000ff', …)]

[Define_Tag: '_unknowntag']
  [Return: (Global: 'excolor_colors')->(find: tag_name)]
[/Define_Tag]
```

blueworld

Unknown Tags
Examples

- Allow data types to be created which support tags whose names are not known at compile time.
- A type could call introspection methods on an XML-RPC server and provide tags for all available remote methods.
- A type could store values by key and allow values to be retrieved using [$Variable->KeyName].
- A type can provide intelligent error messages when an unknown tag is called.

blueworld

Lasso Developer

blueworld

Lasso Developer
Goals

- Allow developers to inspect values in a page while it is running.
- Provide an easy but powerful way to determine what pages to debug.
- Provide feedback about the structure of code which Lasso has parsed.
- Provide a quick way for developers to test code snippets.
- Provide a common interface on Mac OS X / Windows

blueworld

Lasso Developer
Quick Script Window

- Write code snippets and get instant results.
- Use any LDML code including [Inline] ... [/Inline] tags, [Include_URL], etc.
- Store common code snippets for fast retrieval.
- Enhances developer experience by making it easy to test code without placing it in a page.
- Provides instant feedback, allows debugger to be triggered to test custom tags.

blueworld

Lasso Developer
Break Points

- **Specify what pages should activate the debugger.**
- **Uses simple pattern expressions to select multiple pages:**

  *.lasso

  /includes/*.lasso

- **Can select format files loaded in the browser or included files.**
- **Allows the set of pages which are being debugged to be focused or widened as needed.**

blueworld

Lasso Developer
Debugger

- Provides outline view of parsed Lasso code.
- Allows the developer to visually spot errors in tag nesting, logic errors.
- Step command allows LDML statements to be executed one by one.
- Step In command allows custom tags and included files to be stepped into.
- Step Out command triggers execution until the end of the current custom tag or included file.
- Continue command allows execution until the next break point.

blueworld

Lasso Developer
Debugger

- Source code view allows the parsed outline view to be correlated with the source code from the associated format file.
- Expressions allow code to be executed each time an execution step is performed.
- Variables can be watched.
- Tags such as [Error_CurrentError], [Found_Count], [Shown_Count] can be executed.

blueworld

Lasso Developer
LassoApps

- **The code for pages from compiled LassoApps or for custom tags defined within compiled LassoApps cannot be debugged.**
- **This prevents other developers from reverse engineering compiled LassoApps.**

blueworld

Lasso Developer
Road Map

- **Now – Technology preview at Lasso Summit.**
- **Beta – For LPA members by the end of the year.**
- **Release – Early next year.**

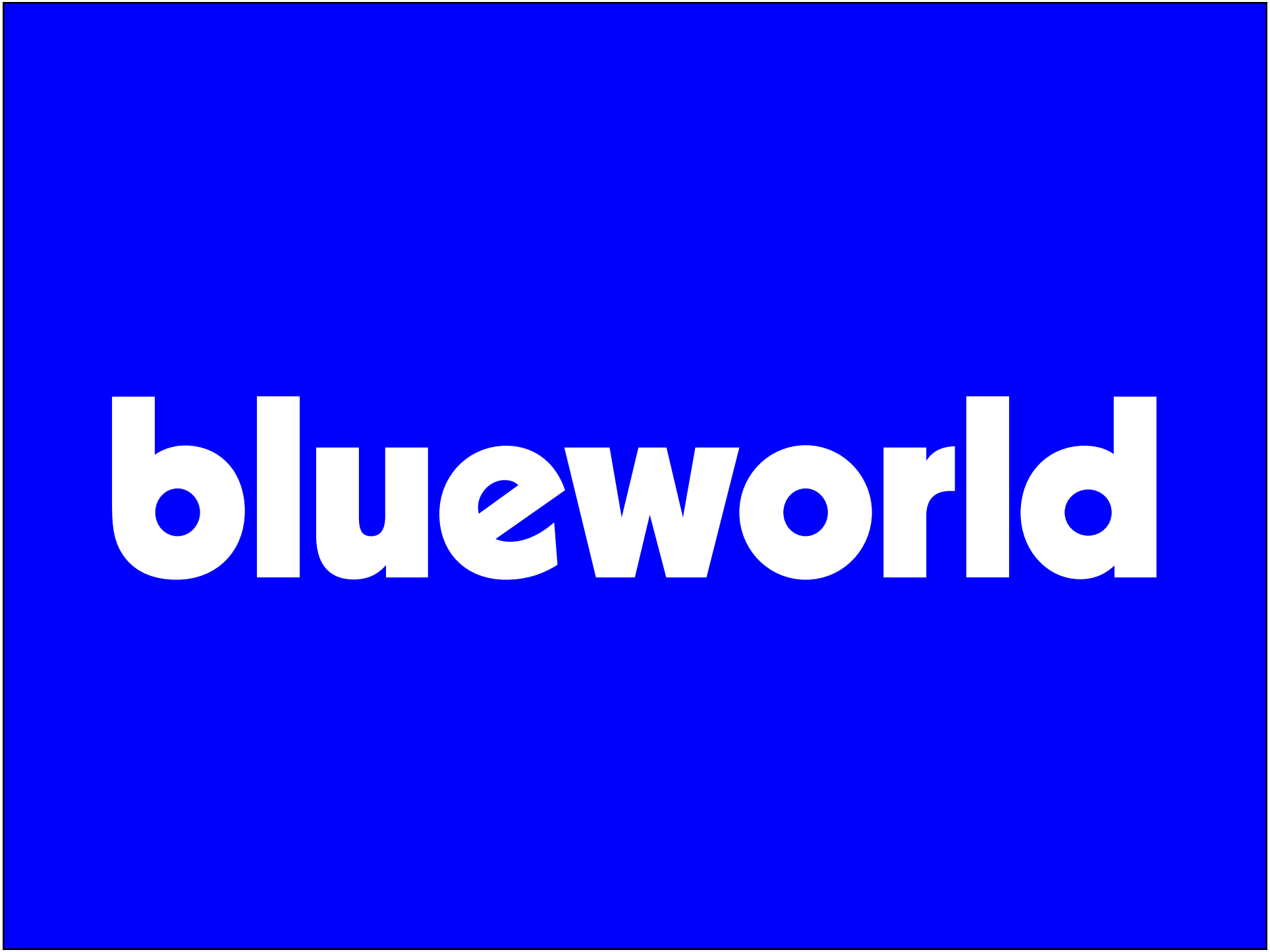
blueworld

Lasso Developer Technology Preview

blueworld

Questions
&
Answers

blueworld


blueworld