let Tuple1 = Tuple<env:IAC, y:object>

let Tuple2 = Tuple<env:IAC, z1:object, z2:object>

let NewTuple1 = Tuple<y:object>

let NewTuple2 = Tuple<z1:object, z2:object>

def foo(): d = new Delegate(methodof(foo))

.method object foo()

t = new Tuple1 t = new NewTuple1

Uninitialize(t)

t.env = new FtnEnvDict<Tuple1>(t)

f = new CodeContext(parent-context, t.env) f = XxxOps.CreateLocalScope<Tuple1>(t, null)

y = 1 t.y = 1

def bar(): c = new Closure(f, constants(bar))  
 d = new Delegate(c, methodof(bar))

.method object bar(Closure closure)

t = new Tuple2 t = new NewTuple2

Uninitialize(t)

t.env = new FtnEnvDict<Tuple2>(t)

f = new CodeContext(parent-context, t.env) f = XxxOps.CreateLocalScope<Tuple1>(t, closure)

z1 = 2 t.z1 = 2

z2 = 3 t.z2 = 3

def baz(): c = new Closure(f, constants(baz))

d = new Delegate(c, methodof(baz))

.method object baz(Closure closure)

t\_parent = closure.Context.Scope.Dict.TupleData t\_parent = (NewTuple2)closure.Scope.Storage

t\_parent\_parent = closure.Context.Scope.Parent.Dict.TupleData t\_parent\_parent = (NewTuple1)closure.Scope.Parent.Storage

print y print t\_parent\_parent.y

print z1 print t\_parent.z1

print z2 print t\_parent.z2

end

end

end

Scope(MethodInfo{ tuple : `TTuple, closure : LocalScope -> currentScope : LocalScope, where `TTuple : Tuple} localScopeFactory, body : IList<Expression>)

internal class GlobalScopeContext {

LanguageContext { get; }

Scope GlobalScope { get; }

}

public abstract class LocalScope {

readonly LocalScope \_parent;

readonly GlobalScopeContext \_global;

LocalScope Parent { get; }

Scope Scope { get { return \_global.Scope; }

LanguageContext Language { get { return \_global.Language; }

abstract object Storage { get; }

# top-level scope:

LocalScope(Scope! globalScope, LanguageContext! language) { \_parent = null; \_global = new GlobalScopeContext(globalScope, langauge); }

# method frame scope:

LocalScope(LocalScope! parent) { \_parent = parent; \_globalScopeInfo = parent.\_global; }

}

# default scope class (used by default factory):

public class TupleDictionaryLocalScope<TTuple> : LocalScope where TTuple : Tuple {

readonly FunctionEnvironmentDictionary<TTuple>! \_environment;

override object Storage { get { return \_environment.TupleData; } }

FunctionEnvironmentDictionary<TTuple>! Environment { get; }

TupleDictionaryLocalScope(FunctionEnvironmentDictionary<TTuple>! env, LocalScope! parent) : base(parent) { \_environment = env; }

}

public sealed class RubyLocalScope<TTuple> : TupleDictionaryLocalScope<TTuple> {

# Ruby specific stuff

}

--- factories ---

# default factory (used when not specified in ScopeExpression):

DlrOps.CreateLocalScope<TTuple>(TTuple! tuple, LocalScope! parent) : LocalScope! {  
 return new TupleDictionaryLocalScope<TTuple>(new FunctionDictionaryEnvironment<TTuple>(tuple), parent);  
}

# Python factory (used when not specified in ScopeExpression):

PythonOps.CreateLocalScope<TTuple>(TTuple! tuple, LocalScope! parent) : LocalScope! {  
 return DlrOps.CreateLocalScope<TTuple>(tuple, parent);  
}

# Ruby factory (used when not specified in ScopeExpression):

RubyOps.CreateLocalScope<TTuple>(TTuple! tuple, LocalScope! parent) : LocalScope! {  
 return new RubyLocalScope<TTuple>(tuple, parent);  
}

Ruby

let Tuple1 = Tuple<env:IAC, x:object>

def foo

.method foo()

t1 = new Tuple1

f1 = RubyOps.CreateMethodScope<Tuple1>(t1, null) -> new RubyLocalScope(t1, null)

x = 1 t1.x = 1

eval(" d = new Delegate(new Closure(f1, CP))

.method #eval1(closure)

f2 = closure.Scope

y = 2 f2.SetVar(‘y’, 2)

")

eval(" d = new Delegate(new Closure(f1, CP))

.method #eval2(closure)

f3 = closure.Scope

z = 3 t3.SetVar(‘z’, 3)

1.times { d = new Delegate(new Closure(f3, CP))

.method #block1(closure)

f3 = closure.Scope

t\_parent = closure.Scope.GetStorage<Tuple1>()

puts x,y,z puts t\_parent.x, f3.GetVar(‘y’), f3.GetVar(‘z’)

}

")

eval("puts x,y,z")

end

Top-level scope, globals

Non-hosted: Could just use Ast.Scope.

Hosted:

* Metadata => define locals for variables defined in metadata + locals that are defined (assigned) in the code.
* Like Python?

JScript

function foo() { .method foo()

$tuple = new Tuple(…)

$frame = JSOps.CreateMethodScope<TTuple>($tuple, <names>, null)

with (obj) { $scope = JSOps.CreateWithBlockScope<TTuple>(null, null, $frame)

JSOps.SetWithObject($scope, obj);

eval(…)

}

}

function bar() { .method bar()

$tuple = new Tuple(…)

$frame = JSOps.CreateMethodScope<TTuple>($tuple, <names>, null)

try {

…

}

catch(e) { $scope = JSOps.CreateCatchBlockScope<TTuple>(null, null, $frame)

eval(…)

}

}

Python

Top-level scope, globals

No local variables on this level, no Ast.ScopeExpression used.

All references to Python globals are resolved by Python, not DLR, and go thru $globals object, which is the global Scope’s dictionary.

TGlobals = OptimizedModule/host provided type (metadata). For now, it could be OptimizedModule/IAC.

# How to pass the scope from compiled code to here? Via $closure?

TGlobals $globals = PythonOps.GetGlobals($closure.Scope)

# metadata available:

$globals.x = 1

# no-metadata, optimized module:

$globals.x.CurrentValue = 1