



# DelphiLive!

The Delphi Developer Conference

Barry Kelly | Embarcadero

# Delphi Compiler RTTI Enhancements

# Existing RTTI

- TypInfo – general type info
  - Flat API focused on properties
  - Extra info via low-level types, unions, pointers
  - Incomplete: metaclasses, records
- IntfInfo – interface type info
  - Optimized and focused on SOAP
  - Relatively low-level
  - Opt-in – not ubiquitous

# Existing RTTI

- ObjAuto – method type info
  - Flat API focused on methods
  - Invocation using variants
  - Callback implementation using variants
  - Extra info via low-level types, unions, pointers
  - Optimized for COM scripting engine support
  - Opt-in – not ubiquitous

# New unit: RTTI.pas

- Object-oriented API
  - Roughly isomorphic with .NET / Java reflection
  - Unification
  - Discoverable
  - Type safe
- Attributes
- Type rooting
- Field info

# Scenarios

- Attributes
  - Metaprogramming
- Dependency Injection
  - Container instantiates
  - Provides constructor arguments
- Object graph tracing
  - Not including stack

# Scenarios

- Serialization
- Debugging
- Scripting
- Documentation / class browser
- Testing
- Remoting

# Scope and Costs

- Public+ methods and properties
- All fields
- All linked-in interface-section types are rooted
  - Linked-in implementation-section
- Extra RTTI is small % increase
- Bigger effect is enforced linking of methods
  - Public method RTTI links in code
  - Dynamic invocation => less visibility

# Reflection Use Case

- Query
  - Search package / type for elements of interest
  - Many temporary elements created and dismissed
- Retain
  - If element (property, method, field etc.) is not to be used immediately, don't want to keep all objects alive
- Use
  - Perform operations using elements

# Query Pattern

```
for type in package.GetTypes
    for method in type.GetMethods
        if method.HasAttribute(...)
then
```

# Object Pooling: TRttiContext

- Most objects are not needed for long
  - Manual lifetime management would be tedious
- Lots of cycles – interfaces not great
  - Interfaces also lack version flexibility
- Object pool strategy
  - Queries are rooted from a TRttiContext object
- The context:
  - Maps underlying RTTI handles to instances
  - Frees instances when destroyed

# RTTI Descriptors

- RTTI objects live in a graph
  - TRttiMethod has declaring type and parameters
  - TRttiParameter has parameter type and owner
  - Cycles
- Descriptor  $\Leftrightarrow$  RTTI object instance
  - Descriptor not bound to a context
  - Migration of instance from one context to another
  - Caching only interesting RTTI instances for later

# Attributes

- Same syntax as Delphi for .NET
  - All attributes descend from TCustomAttribute
- Only simple types allowed in constructors
  - Ordinal types – integers, characters, enums
  - Strings
  - Type references
- RTTI objects have GetAttributes method
  - Attributes are owned by the RTTI object, and therefore implicitly part of the context

# Method Invocation

- TRttiMethod.Invoke
  - Invoke instance, class and class static methods
- TRttiConstructor.Invoke
  - Dynamically construct instances without needing virtual constructors and metaclasses
  - Required for general custom attribute support
- How to handle argument values?
  - ObjAuto uses Variant – has limitations

# TValue - a simple top type

- Is a tuple of raw value data and type info
- Does not support operators, methods, etc.
  - Not a replacement for Variant
- Conversions in:
  - Has implicit conversions where possible
  - Has explicit generic conversion for other types
- Conversions out:
  - Runtime typed with explicit type (generic)
  - Untyped access to underlying bytes

# TRttiContext

```
function GetTypes: TArray<TRttiType>
```

```
function GetType(TypeInfo): TRttiType
```

```
// Maybe:
```

```
function GetType<T>: TRttiType
```

```
// GetPackages; types scoped to package
```

# TRttiType

```
function GetMethods: TArray<TRttiMethod>
function GetProperties // TRttiProperty
function GetFields: TArray<TRttiField>
property IsManaged: Boolean
property TypeSize: Integer
function GetMethod(Name: string)
// etc.; subtypes have more info
```

# TRttiProperty

```
property Name: string
```

```
property PropertyType: TRttiType
```

```
function GetValue(Obj: TObject): TValue
```

```
procedure SetValue(Obj: TObject; V: TValue)
```

# TRttiMethod

```
function GetParameters:
```

```
    TArray<TRttiParameter>
```

```
property ReturnType: TRttiType
```

```
function Invoke(
```

```
    const Args: TArray<TValue>): TValue
```

```
function Invoke(
```

```
    const Args: array of TValue): TValue
```

# Method Invocation

```
method: TRttiMethod;  
  
// ...  
  
method.Invoke(instance,  
    [42, 'foo',  
        TValue.From(wsNormal)]);
```

# TRttiField

```
property FieldType: TRttiType
```

```
function GetValue(Obj: TObject): TValue
```

```
procedure SetValue(Obj: TObject;  
    Value: TValue)
```

```
property Offset: Integer
```

# Demo: Reflection Explorer

# Demo: Attribute Usage

# Demo: Heap Tracing