

Metadata

Tomáš Matoušek

tm.matfyz.cz/teaching/cclr.htm

Overview

- data describing managed data and code
- stored together with code and data in one file
- allow for reflection
 - a technique for reading metadata at run-time
- custom metadata
- metadata APIs
 - managed
 - BCL: System.Reflection, System.Reflection.Emit
 - PERWAPI
 - Reflector API
 - unmanaged
 - CLR COM API

Assemblies

- assembly
 - deployment unit
 - set of one or more files
 - described by manifest (content + additional info)
- files logically contained in an assembly
 - modules (contain IL code)
 - resources
 - other files
- primary module
 - module containing manifest
 - assembly contains at most one primary module
- modules have PE/COFF structure
 - .text section contains IL code, managed resources, metadata

Metadata Organization

- metadata of each module stored in streams
 - heaps and serialized tables
- heaps
 - contiguous sequences of items
 - types:
 - string heap – zero terminated UTF-8 encoded strings
 - GUID heap – 16B long GUIDs
 - blob heap – binary data, each datum preceded by its length
- tables
 - form a relational schema
 - entities: types, methods, fields, parameters, ...

Streams

name	type	content description
#Strings	string heap	names of language elements (types, fields, ...)
#Blob	blob heap	signatures, default values for fields and params, ...
#GUID	GUID heap	GUIDs (each module has a GUID assigned)
#US	blob heap	Unicode string literals (aka user strings)
#~	tables	optimized metadata tables
#-	tables	raw metadata tables

- a persistent module contains the #~ stream (tables are optimized)
- a dynamic module contains the #- stream (tables are not optimized)

Schema of Metadata Tables

- hardcoded in CLR
- defines columns of each table (type, order and count)
- each table identified by a number (0...63)
- column types:
 - RID<tableId> – an index of a record in explicitly specified table
 - token – a pair [tableId, index]
 - offset in #Strings, #Blob or #GUID stream
 - integer number type
- RIDs used internally
- tokens are parts of instructions and used by metadata APIs
 - e.g. a part of a **call** instruction is token of the method to be called

Metadata Entity Hierarchies

- parent-children lists
 - a list of all fields in a type
 - a list of all methods in a type
 - a list of all parameters of a method
- optimized tables (#~ stream)
 - children tables are sorted by their parent row index
 - parent entity points to the first child
 - the last child determined by the next parent's first child
- dynamic tables (#- stream)
 - edit-and-continue scenario
 - intermediate indexing table

Table Name Suffixes

- -Def tables
 - entities defined in a module containing the table
- -Ref tables
 - entities referred in the module but defined in another one
- -Ptr tables
 - indexing tables (only in #~ stream)

Metadata Tables

name	contains
Module	module name, GUID
Assembly	information contained in the manifest <ul style="list-style-type: none">• a single record, only in the primary module
AssemblyRef	a list of referenced assemblies (strong names)
ManifestResource	managed resources contained in the module
TypeDef	types defined in the module
FieldDef	fields defined in the module
MethodDef	methods defined in the module
Signature	signatures
CustomAttribute	user defined metadata
...	entire schema comprises of 44 tables ...

Adding Custom Metadata

- **problem:**
 - we want to add a new column to an existing metadata table
 - but tables can't be changed (schema is hardcoded)
- **solution:**
 - CustomAttribute table contains custom metadata
 - takes advantage of tokens:
 - token can reference items in other tables
 - each row is bound to some table

CustomAttribute Table

- consists of three columns:
 - parent (token)
 - metadata to which the attribute is assigned
 - refers to a table: Assembly, TypeDef, MethodDef, ...
 - type (token)
 - defines the attribute constructor
 - refers to Method or MemberRef table
 - value (offset in #Blob stream)
 - serialized data
 - constructor's arguments (positional parameters)
 - name-value pairs (named parameters)

Pseudo-custom Attributes

- not stored in CustomAttribute table
- existing metadata items modified instead
- however, used with the same syntax as CAs
- examples
 - DllImportAttribute (on methods)
 - SerializableAttribute (on types)
 - NonSerializedAttribute (on fields)
 - transient keyword in Java

Inspecting Metadata

- via unmanaged APIs
 - low level – working directly with metadata tables
 - used by compilers and tools (ILDASM)
- via System.Reflection classes in BCL
 - higher level of abstraction
 - classes representing types, methods, fields, ...
 - uses runtime metadata representation (recall **EEClass**)
 - some structures created by loader
 - other structures created lazily when requested for the first time
 - no access to IL before FW 2.0
- via other APIs
 - PEAPI, PERWAPI, Reflector API

runtime*.cs
typehandle.h
field.h
method.hpp

Runtime Metadata Representation

- structures representing types, fields and methods respectively
 - System.RuntimeFieldHandle
 - holds a pointer to **FieldDesc**
 - System.RuntimeMethodHandle
 - holds a pointer to **MethodDesc**
 - System.RuntimeTypeHandle
 - mapped to **TypeHandle**
 - holds a union of
 - pointer to **MethodTable**
 - if types don't share **MethodTables**
 - pointer to **TypeDesc**
 - parameterized types, e.g. arrays, pointers (**ParamTypeDesc**)
 - function types (**FunctionTypeDesc**)
 - two types are equal iff the union values are equal

ldtoken IL instruction

- loads a runtime representation of an entity identified by a token
 - result is a `Runtime{Type|Method|Field}Handle`
- token is a part of the instruction
 - represents an entity defined or referenced by the current assembly
- implementation of the C# operator `typeof(T)`
 - C# compiler emits
 - a `ldtoken` instruction with the token of `T`
 - a call instruction with the token of `Type::GetTypeFromHandle()` method

```
ldtoken <type token>
```

```
call class Type::GetTypeFromHandle(valueType RuntimeTypeHandle)
```

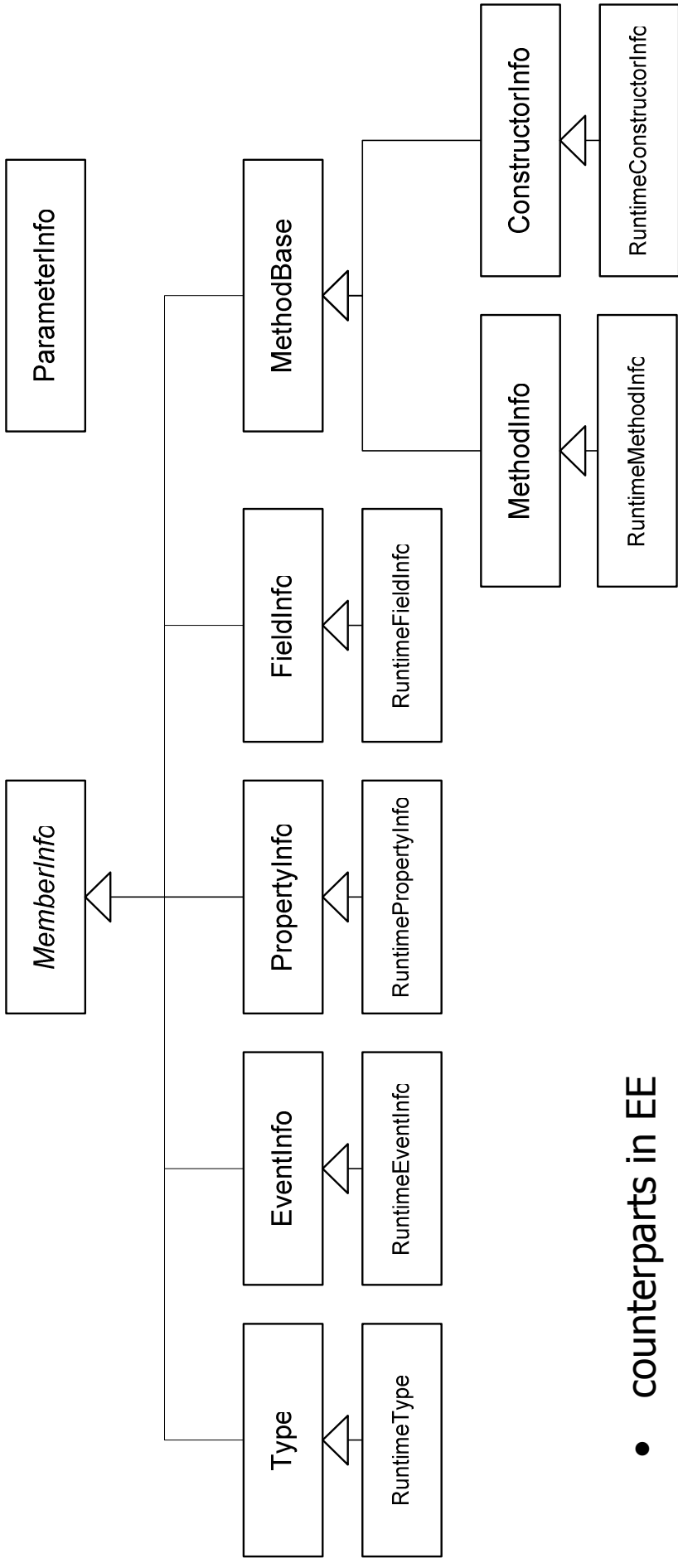
- JIT (`ldtoken`)
 - looks-up a metadata table using the token as an “address”
 - emits the value of `RuntimeTypeHandle`
- `Type.GetTypeFromHandle()` → `RuntimeType.GetTypeFromHandleImpl()`
 - returns instance of `RuntimeType`, a subclass of `Type`

object.h

*Info.cs

Runtime*Info.cs

Managed Reflection Classes



- counterparts in EE
 - MemberInfo
 - **BaseObjectWithCachedData : Object**
 - holds a pointer to Reflection Cache

EE Reflection Classes (version 1.* only)

- holds all metadata information
 - used by Reflection API
 - hierarchy:
 - **ReflectBase**
 - **ReflectField**
 - **ReflectMethod**
 - **ReflectProperty**
 - **ReflectEvent**
 - **ReflectClass**
 - **ReflectBaseClass**
 - **ReflectArrayClass**
 - contain all information stored in metadata
 - joined with RuntimeInfos via a pointer
 - all RuntimeInfos mapped to ReflectBaseObject in EE
 - contain pointer to an instance of ReflectXxx class
- metadata stored in RuntimeInfos since version 2.0