Notes - 8/19: SENT

Wednesday, September 07, 2011 10:19 AM

es5.istr

dom.istr

winrt.istr

jQuery.istr

lib.istr

```
//////////////
//15.1 - Global Object
//////////////
extern NaN: number;
extern Infinity: number;
extern undefined: any;
extern eval(x: string): any;
extern parseInt: { (string: string): number, (string: string, radix: string): number };  // DECISION: should be
able to write as two declarations of overload
extern parseFloat(string: string): number; // DECISION: extern function should be legal
extern isNaN(number: number): bool;
extern isFinite(number: number): bool;
extern decodeURI(encodedURI: string): string;
extern decodeURIComponent(encodedURIComponent: string): string;
extern encodeURI(uri: string): string;
extern encodeURIComponent(uriComponent: string): string;

// Helper interfaces needed for PropertyDEscriptor typing
interface IPropertyDescriptor {
  configurable: bool;
  enumerable: bool;

  value: any;
  writable: bool;

  get(): any;
  set(v: any): void;
}

interface IPropertyDescriptorMap {
  // TODO: Is this legit?
  // DECISION: yes.
  [s:string]: IPropertyDescriptor;
}

// DECISION: Indexing into arrays is only by number
// DECISION: can provide [x: number] and [x: string]
// DECISION: [x: number] is statically preferred for number

// T[] <===> { [x: number]: T; length: number; }

//foo.bar
//foo["bar"]

// DECISION: indexing expression is allowed on any, and on statically typed things that support []:foo
// TOPIC - let's try to figure out generics in a future meeting


// var x: number = 5;
// var x: Number = Object(5);
// (5).toString();



//////////////
//15.2 - Object Objects
//////////////
extern class Object() {
      // TODO:  This is presumably the base class of all classes with undeclared base
      // TODO:  It also seems to be the corresponding type for runtime things typed as Object
    // TODO:  There is potentially a hole here though that Object would mean it has
    //       Object.prototype in it's protot chain, but ToObject does not guarantee that

 // DECISION: Yes on above.
 // DECISION: below...
 // Object -> {}
 // number -> {}
 // Number -> Object


 public toString(): string;
 public toLocaleString(): string;
 public valueOf(): Object;
 public hasOwnProperty(v: string): bool;
 public isPrototypeOf(v: Object): bool;
 public propertyIsEnumerable(v: string): bool;

 // TODO: .constructor? I assume is implict in class definition,
 //       and has an anonymous interface type corresponding to the Object module

}
module Object {
 public new(): Object;
 public new(value: any): Object;
 public (): Object;
 public (value: any): Object;

 // TODO: Is there a .prototype property?
 //      I believe that is implict in the definition Object as a
 //      class, and it's value is actually "any"

 // TODO: There is a .length

 // TODO: Is this typing too strict - are object literals instances of Object?
 public getPrototypeOf(o: Object): Object;
 public getOwnPropertyDescriptor(o: Object, p: string): IPropertyDescriptor;
 public getOwnPropertyNames(o: Object): string[];
 public create(o: Object): Object;
 public create(o: Object, properties: IPropertyDescriptorMap): Object;
 public defineProperty(o: Object, p: string, attributes: IPropertyDescriptor): Object;
 public defineProperties(o: Object, properties: IPropertyDescriptorMap): Object;
 public seal(o: Object): Object;
 public freeze(o: Object): Object;
 public preventExtensions(o: Object): Object;
 public isSealed(o: Object): bool;
 public isFrozen(o: Object): bool;
 public isExtensible(o: Object): bool;
 public keys(o: Object): string[];
}

//////////////
//15.3 - Function Objects
//////////////
extern class Function() {
 // TODO: All function literals should in principle be instances of this class
 //       The have length, etc.

 public toString(): string; // TODO: Is this needed in class signture, given that it just overrides?
 public apply(thisArg: any, argArray: any[]): any;
 public call(thisArg: any, ...argArray: any[]): any;
 public bind(thisArg: any, ...argArray; any[]): Function; // TODO: Not sure about this

 public length: number

 // TODO: There is also an override of the internal [HasInstance], capturing that may be useful for 'is'?
}
module Function {
 //  TODO: The Function constructor object is actually itself a Function
 //        I don't believe we can capture that  (Function instanceof Function)


 public new(...args: string[]): Function
 public (...args: string[]): Function

 public length: number;
}

//////////////
//15.4 - Array Objects
//////////////

//////////////
//15.5 - String Objects
//////////////

//////////////
//15.6 - Boolean Objects
//////////////

//////////////
//15.7 - Number Objects
//////////////

//////////////
//15.8 - Math Objects
//////////////


class Foo() {
}
Foo.interface === { new() : Foo }
var foo = new Foo();
var x : Foo.interface = foo.constructor;


//////////////
//15.9 - Date Objects
//////////////
extern class Date(value: number) {
 // TODO: .constructor? I assume is implict in class definition, and has an anonymous interface type
corresponding to the Date module
 //public constructor: Date.interface;
 // DECISION: don't have to declare the above

 public toString(): string;
 public toDateString(): string;
 public toTimeString(): string;
 public toLocaleString(): string;
 public toLocaleDateString(): string;
 public toLocaleTimeString(): string;
 public valueOf(): number;
 public getTime(): number;
 public getFullYear(): number;
 public getUTCFullYear(): number;
 public getMonth(): number;
 public getUTCMonth(): number;
 public getDate(): number;
 public getUTCDate(): number;
 public getDay(): number;
 public getUTCDay(): number;
 public getHours(): number;
 public getUTCHours(): number;
 public getMinutes(): number;
 public getUTCMinutes(): number;
 public getSeconds(): number;
 public getUTCSeconds(): number;
 public getMilliseconds(): number;
 public getUTCMilliseconds(): number;
 public getTimezoneOffset(): number;
 public setTime(time: number);
 public setMilliseconds(ms: number);
 public setUTCMilliseconds(ms: number);
 public setSeconds(): number;
 public setUTCSeconds(): number;
 public setMinutes(): number;
 public setUTCMinutes(): number;
 public setHours(): number;
 public setUTCHours(): number;
 public setDate(): number;
 public setUTCDate(): number;
 public setMonth(): number;
 public setUTCMonth(): number;
 public setFullYear(): number;
 public setUTCFullYear(): number;
 public toUTCString(): string;
 public toISOString(): string;
 public toJSON(): string;

}
extern module Date {
 // Called as a function
 public (): string;

 // Called as a constructor
 public new() : Date;
 public new(year: number, month: number): Date;
 public new(year: number, month: number, date: number): Date;
 public new(year: number, month: number, date: number, hours: number): Date;
 public new(year: number, month: number, date: number, hours: number, minutes: number): Date;
 public new(year: number, month: number, date: number, hours: number, minutes: number, seconds:
number): Date;
 public new(year: number, month: number, date: number, hours: number, minutes: number, seconds:
number, ms: number): Date;
 // TODO: Does the "implementation" constructor signature need to be repeated here - I think so
 // DECISION: yes.
 public new(value: number): Date;

 // TODO: Is there a .prototype property?  I believe that is implict in the definition Date as a class, and
it's value is actually "any"
 // DECISION: Yes, not sure whether this needs to be on instances.

 // DECISION: The below should be legal (string: string)
 public parse(string: string): number;
 public UTC(year: number, month: number): number;
  public UTC(year: number, month: number, date: number): number;
  public UTC(year: number, month: number, date: number, hours: number): number;
  public UTC(year: number, month: number, date: number, hours: number, minutes: number): number;
  public UTC(year: number, month: number, date: number, hours: number, minutes: number, seconds:
number): number;
  public UTC(year: number, month: number, date: number, hours: number, minutes: number, seconds:
number, ms: number): number;
  public now(): number;

}

//////////////
//15.10 - RegExp Objects
//////////////

//////////////
//15.11 - Error Objects
//////////////

//////////////
//15.12 - JSON Object
//////////////
```