

```
1  # Perl binding to API given in http://www.w3.org/TR/2008/WD-DDR-Simple-API-20080404/
2  # (c) Rotan Hanrahan 2008 under authority from MobileAware
3  # This code is not guaranteed fit for any purpose whatsoever. Use at your own risk.
4
5  # Core Vocabulary: http://www.w3.org/TR/2008/NOTE-ddr-core-vocabulary-20080414/
6  # Core Vocabulary IRI: http://www.w3.org/2008/01/ddr-core-vocabulary
7
8
9  #####
10 ## Part 1 #
11 ## This section shows how a Perl programmer would use the DDR Simple API #
12 ## Demonstrates: #
13 ##   - Use of Service methods to interact with DDR. #
14 ##   - Getting single property value and collections of property values. #
15 ##   - Use of property term names and aspect names. #
16 ##   - Catching exceptions thrown by the DDR. #
17 #####
18
19 eval { # TRY
20     local $SIG{'__DIE__'};
21
22     # Instantiate new Service
23     $ss = Service->new();
24     $ss->initialize('http://www.w3.org/2008/01/ddr-core-vocabulary',undef);
25
26     print 'Using API (' . $ss->getImplementationVersion() . ') with Repository (' . $ss->getDataVersion() . ")\n";
27
28     # Populate new instance of Evidence
29     my $e = $ss->newHTTPEvidence(); #my $e = Evidence->new();
30     $e->put('User-Agent','Mozthing1.2e (X11; en-US; v12)');
31     $e->put('Accept','text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*');
32
33     print "Evidence from the delivery context:\n";
34     print '  User-Agent: ' . $e->get('User-Agent') . "\n";
35     print '  Accept: ' . $e->get('Accept') . "\n";
36
37     # Names of the aspects in which we are interested (aspect IRI is same as property IRI)
38     my $softwareAspect = 'webBrowser';
39     my $hardwareAspect = 'device';
```

```

40
41 # Two display properties of interest: Height and Width
42 my $heightPropName = $ss->newPropertyName('displayHeight');
43 my $widthPropName = $ss->newPropertyName('displayWidth');
44
45 # The two properties in the software aspect
46 my $heightPropRefSW = $ss->newPropertyRef($heightPropName,$softwareAspect);
47 my $widthPropRefSW = $ss->newPropertyRef($widthPropName,$softwareAspect);
48 my $propRefArraySW = [ $heightPropRefSW, $widthPropRefSW ];
49
50 # The two properties in the hardware aspect
51 my $heightPropRefHW = $ss->newPropertyRef($heightPropName,$hardwareAspect);
52 my $widthPropRefHW = $ss->newPropertyRef($widthPropName,$hardwareAspect);
53 my $propRefArrayHW = [ $heightPropRefHW, $widthPropRefHW ];
54
55 # Get the values for these properties in this aspect given this evidence
56 print "Getting specific software values for this context:\n";
57 my $spvsSW = $ss->getPropertyValues($e,$propRefArraySW);
58 print "  Browser Display Height : " . $spvsSW->getValue($heightPropRefSW)->getInteger() . "\n";
59 print "  Browser Display Width  : " . $spvsSW->getValue($widthPropRefSW)->getInteger() . "\n";
60
61 # Get the values for these properties in this aspect given this evidence
62 # This demonstrates exception catching for case where one or both properties are not available
63 eval { # TRY
64     local $SIG{'__DIE__'};
65     print "Getting specific hardware values for this context:\n";
66     my $spvsHW = $ss->getPropertyValues($e,$propRefArrayHW);
67     print "  Physical Display Height : " . $spvsHW->getValue($heightPropRefHW)->getInteger() . "\n";
68     print "  Physical Display Width  : " . $spvsHW->getValue($widthPropRefHW)->getInteger() . "\n";
69 };
70 if ($?) { # CATCH
71     if ($?-isa('NameException')) {
72         print ' * * Caught expected NameException: [(' . $?-getCode() . ') ' . $?-getMessage() . "]\n";
73     }
74 }
75
76 # Get all known data
77 print "Getting all values for this context:\n";
78 my $pvsAll = $ss->getPropertyValues($e);

```

```

79  my @allProperties = $pvsAll->getAll();
80  foreach my $pv (@allProperties) {
81      my $pRef = $pv->getPropertyRef();
82      print ' ' . $pRef->getLocalPropertyName() . ' is ' . $pv->getString() . "\n";    # Have to assume String, no way to tell!
83  }
84
85  # Get image support information
86  print "Getting image format support for this context:\n";
87  my $imgFmtPropName = $ss->newPropertyName('imageFormatSupport');
88  my $spv = $ss->getPropertyValue($e,$imgFmtPropName);
89  my @supportedImageFormats = @{$spv->getEnumeration()};
90  foreach my $imgFmt (@supportedImageFormats) {
91      print ' ' . $imgFmt . " is supported\n";
92  }
93
94  # All-in-one: get the physical height of the device
95  print "Getting information via Simple API methods only:\n";
96  print " Device Height = " . $ss->getPropertyValue($e,$ss->newPropertyRef($ss->newPropertyName('displayHeight'),'device'))->
getString() . "\n";
97  print "Getting information via implementation specific constructor:\n";
98  print " Device Height = " . $ss->getPropertyValue($e,PropertyRef->new('displayHeight','device'))->getString() . "\n";
99  # Most likely convenience method we will be asked for:
100  # print " Device Height = " . $ss->getPropertyValue($e,'displayHeight','Device')->getString() . "\n";
101
102  # Using the PropertyRef proposal
103  print "Getting information via PropertyRef:\n";
104  my $spr = PropertyRef->new('displayHeight','webBrowser');
105  print ' Browser height = ' . $ss->getPropertyValue($e,$spr)->getInteger() . "\n";
106
107  };
108  if ($@) { # CATCH
109      local $SIG{'__DIE__'};
110      if ($@->isa('BaseException')) {
111          print 'UNEXPECTED: Caught Exception: [' . $@->getCode() . ' ' . $@->getMessage() . "]\n";
112          die 'Stopping because of unexpected exception';
113      }
114      else {
115          die $@;
116      }

```

```

117 }
118
119 #####
120 ## Part 2 #
121 ## Shows how a typical Perl programmer might wrap the DDR Simple API to make #
122 ## use of the defaults, and pre-populate common parameters for re-use. #
123 ## Two subroutines are defined: convenienceInit() and getPropVal(e,a,p) #
124 #####
125
126 # The way Perl hackers might do this using convenience methods
127 eval { # TRY
128     local $SIG{'__DIE__'};
129     convenienceInit();
130     my $e = $ss->newHTTPEvidence();
131     $e->put('User-Agent','Mozthing1.2e (X11; en-US; v12)');
132     $e->put('Accept','text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*');
133     print "Getting information via custom convenience methods:\n";
134     print " Device Height = " . getPropVal($e,'displayHeight','device')->getInteger() . "\n";
135     print " Device Width = " . getPropVal($e,'displayWidth','device')->getInteger() . "\n";
136     print " Image formats = " . join(', ',getPropVal($e,'imageFormatSupport','webBrowser')->getEnumeration()) . "\n";
137 };
138 if ($?) { # CATCH
139     local $SIG{'__DIE__'};
140     if ($?-isa('BaseException')) {
141         print 'UNEXPECTED: Caught Exception: [' . $?-getCode() . ' ' . $?-getMessage() . "]\n";
142         die 'Stopping because of unexpected exception';
143     }
144     else {
145         die $?;
146     }
147 }
148
149 #####
150 ## Part 3 #
151 ## Standard DDR Simple API tests, derived from the implementation test class #
152 #####
153
154 eval { # TRY
155     local $SIG{'__DIE__'};

```

```

156 print "BEGIN Standard API tests -----\\n";
157 my $s = Service->new();
158 $s->initialize('http://www.w3.org/2008/01/ddr-core-vocabulary',undef);
159 my $e = $s->newHTTPEvidence(); #my $e = Evidence->new();
160 $e->put('User-Agent','FakeDevice');
161 DDRTSimpleAPITest::initTest(
162     $s, $e,
163     'http://www.w3.org/2008/01/ddr-core-vocabulary','webBrowser', 'device', 11,
164     'vendor', 'model', 'device', 'device', 'Acme Co.',
165     'displayWidth', 'device', 120,
166     'cookieSupport', 'webBrowser', 1, # 1 = true
167     'markupSupport', 'webBrowser', @[[ 'xhtmlBasic10', 'xhtmlMP10' ]])
168 );
169 %r = DDRTSimpleAPITest::getReport();
170 foreach $k (sort keys %r) {
171     print " $k = " . $r{$k} . "\\n";
172 }
173 print "END Standard API tests -----\\n";
174 };
175 if ($?) { # CATCH
176     if ($?-isa('BaseException')) {
177         print 'UNEXPECTED: Caught Exception: [' . $?-getCode() . ' ' . $?-getMessage() . ']'\\n";
178         die 'Stopping because of unexpected exception';
179     }
180     else {
181         die $?;
182     }
183 }
184
185
186 exit 1;
187
188
189
190 # These convenience methods might be hidden away in a custom Perl module
191 sub convenienceInit {
192     $ddrSrv = Service->new();
193     $ddrSrv->initialize('http://www.w3.org/2008/01/ddr-core-vocabulary');
194     $DDRNullPropRef = PropertyRef->new('NULL','');

```

```

195     $DDRNullValue = PropertyValue->new($DDRNullPropRef, '000000');
196 }
197 sub getPropVal {
198     my ($ev,$p,$a) = @_;
199     my $result;
200     eval { # TRY
201         local $SIG{'__DIE__'};
202         $result = $ddrSrv->getPropertyValue($ev,$p,$a, 'http://www.w3.org/2008/01/ddr-core-vocabulary');
203     };
204     if ($@) { # CATCH
205         if ($@->isa('BaseException')) {
206             print 'UNEXPECTED: Caught Exception: [' . $@->getCode() . ' ' . $@->getMessage() . "]\n";
207             die 'Stopping because of unexpected exception';
208         }
209         else {
210             die $@;
211         }
212     }
213     if (!$result->exists()) { return $DDRNullValue; }
214     return $result;
215 }
216
217 # ==== END OF USER CODE ====
218
219
220
221
222
223 #####
224 ## Part 4 #
225 ## This is a set of Perl packages that implement the DDR Simple API. #
226 ## Public methods are marked thus in comments: [DDR Simple API] #
227 #####
228
229
230
231 # ==== START OF API PACKAGES ====
232
233 # -----

```

```
234 package Service;
235
236 use Scalar::Util qw(blessed);
237 use Carp;
238
239 # Constructor is not part of Simple API specification
240 sub new {
241     my $pkg = shift;
242     my $this = {};
243     bless $this, $pkg;
244     return $this;
245 }
246
247 # [DDR Simple API] Service public String getImplementationVersion()
248 sub getImplementationVersion {
249     return "1.1.0 http://www.w3.org/TR/2008/WD-DDR-Simple-API-20080404/";
250 }
251
252 # [DDR Simple API] Service public String getDataVersion()
253 sub getDataVersion {
254     my $this = shift;
255     return $this->{REPOSITORY}->getDataVersion();
256 }
257
258 # [DDR Simple API] Service public PropertyRef[] listPropertyRefs()
259 sub listPropertyRefs {
260     my $this = shift;
261     my $repository = $this->{REPOSITORY};
262     my @knownVocabularyIRIs = @{$this->{REPOSITORY}->getVocabularies()};
263     my @propertyRefs = ();
264     for my $vocabIRI (@knownVocabularyIRIs) {
265         my @propertyNames = @{$repository->getPropertyNames($vocabIRI)};
266         for my $propertyName (@propertyNames) {
267             my @supportedAspects = @{$repository->getSupportedAspects($vocabIRI,$propertyName)};
268             for my $aspectName (@supportedAspects) {
269                 push(@propertyRefs,PropertyRef->new(PropertyName->new($propertyName,$vocabIRI),$aspectName));
270             }
271         }
272     }
```

```
273     return \@propertyRefs;
274 }
275
276 # [DDR Simple API] Service public void initialize(String defaultVocabularyIRI, Properties props) throws SystemException; // Vocabulary cannot be 'null'
277 sub initialize {
278     my $this = shift;
279     $this->{DEFAULTVOCABULARY} = shift;
280     if (@_) {
281         my $properties = shift; # In Perl, a Properties object is typically a hash
282         if (defined $properties) {
283             $this->{PROPS} = \$properties;
284         }
285     }
286     $this->{REPOSITORY} = CustomDDRImplementation->new();
287 }
288
289 # [DDR Simple API] Service public PropertyValues getPropertyValues(Evidence evidence)
290 # [DDR Simple API] Service public PropertyValues getPropertyValues(Evidence evidence, String localAspectName)
291 # [DDR Simple API] Service public PropertyValues getPropertyValues(Evidence evidence, String localAspectName, String vocabularyIRI)
292 # [DDR Simple API] Service public PropertyValues getPropertyValues(Evidence evidence, PropertyRef[] propertyRefs)
293 sub getPropertyValues {
294     my $this = shift;
295     my $paramCount = @_;
296     my ($p1,$p2,$p3) = @_;
297     if ($paramCount == 1 && $p1->isa('Evidence')) { # getPropertyValues(Evidence)
298         return _getPropertyValues_Evidence($this,$p1);
299     }
300     if ($paramCount == 2) {
301         if ($p1->isa('Evidence') && !blessed($p2) && ref($p2) eq 'ARRAY') { # getPropertyValues(Evidence, PropertyRef[])
302             return _getPropertyValues_Evidence_ARRAY($this,$p1,$p2);
303         }
304         else { # getPropertyValues(Evidence, String)
305             return _getPropertyValues_Evidence_LocalAspectName($this,$p1,$p2);
306         }
307     }
308     if ($paramCount == 3) { # getPropertyValues(Evidence, String, String)
309         return _getPropertyValues_Evidence_LocalAspectName_AspectIRI($this,$p1,$p2,$p3);
310     }
311     die('Method signature unknown');
```

```
312 }
313
314 sub _getPropertyValues_Evidence {
315     my $this      = shift;
316     my $evidence   = shift;
317     my $pvs = PropertyValues->new();
318     my @knownVocabularyIRIs = @{$this->{REPOSITORY}->getVocabularies()};
319     foreach my $vocabularyIRI (@knownVocabularyIRIs) {
320         my @localPropertyNames = @{$this->{REPOSITORY}->getPropertyNames($vocabularyIRI)};
321         foreach my $localPropertyName (@localPropertyNames) {
322             my $defaultAspectForProperty = $this->{REPOSITORY}->getDefaultAspect($localPropertyName,$vocabularyIRI);
323             my $propertyName = $this->newPropertyName($localPropertyName);
324             my $propertyRef = $this->newPropertyRef($propertyName,$defaultAspectForProperty);
325             my $actualPropertyValue = $this->{REPOSITORY}->getValue($evidence,$localPropertyName,$defaultAspectForProperty,
326 $vocabularyIRI);
327             if (defined $actualPropertyValue) {
328                 my $propertyValue = PropertyValue->new($propertyRef,$actualPropertyValue);
329                 $pvs->setValue($propertyRef,$propertyValue);
330             }
331         }
332     }
333     return $pvs;
334 }
335
336 sub _getPropertyValues_Evidence_ARRAY {
337     my $this      = shift;
338     my $evidence   = shift;
339     my $propertyRefARRAY = shift;
340     my $vocabularyIRI = $this->{DEFAULTVOCABULARY};
341     my $pvs = PropertyValues->new();
342     foreach my $pRef (@{$propertyRefARRAY}) {
343         if (!$pRef->isa('PropertyRef')) {
344             die SystemException->new('PropertyRef[] contained a non-PropertyRef element');
345         }
346         my $localPropertyName = $pRef->getLocalPropertyName();
347         my $aspect = $pRef->getAspectName();
348         if (!defined($aspect) || $aspect eq '') {
349             my $aspect = $this->{REPOSITORY}->getDefaultAspect($localPropertyName,$vocabularyIRI);
```

```
350     my $namespace = $pRef->getNamespace();
351     my $actualPropertyValue = $this->{REPOSITORY}->getValue($evidence,$localPropertyName,$aspect,$namespace);
352     if (defined $actualPropertyValue) {
353         my $propertyValue = PropertyValue->new($pRef,$actualPropertyValue);
354         $pvs->setValue($pRef,$propertyValue);
355     }
356 }
357 return $pvs;
358 }
359
360 sub _getPropertyValues_Evidence_LocalAspectName { # gets all values
361     my $this      = shift;
362     my $evidence   = shift;
363     my $localAspectName = shift;
364     my $aspectIRI  = $this->{DEFAULTVOCABULARY}; # In Simple API, aspect IRI defaults to the property IRI
365     return $this->_getPropertyValues_Evidence_LocalAspectName_AspectIRI($evidence,$localAspectName,$aspectIRI);
366 }
367
368 sub _getPropertyValues_Evidence_LocalAspectName_AspectIRI { # gets all values
369     my $this      = shift;
370     my $evidence   = shift;
371     my $localAspectName = shift;
372     my $aspectIRI  = shift;
373
374     my $pvs = PropertyValue->new();
375     my @knownVocabularyIRIs = @{$this->{REPOSITORY}->getVocabularies()};
376     foreach my $vocabularyIRI (@knownVocabularyIRIs) {
377         my @localPropertyNames = @{$this->{REPOSITORY}->getPropertyNames($vocabularyIRI)};
378         foreach my $localPropertyName (@localPropertyNames) {
379             my $defaultAspectForProperty = $this->{REPOSITORY}->getDefaultAspect($localPropertyName,$vocabularyIRI);
380             my $propertyName = $this->newPropertyName($localPropertyName);
381             my $propertyRef = $this->newPropertyRef($propertyName,$defaultAspectForProperty);
382             my $actualPropertyValue = $this->{REPOSITORY}->getValue($evidence,$localPropertyName,$defaultAspectForProperty,
383 $vocabularyIRI);
384             if (defined $actualPropertyValue) {
385                 my $propertyValue = PropertyValue->new($propertyRef,$actualPropertyValue);
386                 $pvs->setValue($propertyRef,$propertyValue);
387             }
388         }
389     }
390 }
```

```
388     }
389     return $pvs;
390
391 }
392
393 # [DDR Simple API] Service public PropertyValue getPropertyValue(Evidence evidence, PropertyRef propertyRef) throws NameException, SystemException;
394 # [DDR Simple API] Service public PropertyValue getPropertyValue(Evidence evidence, PropertyName propertyName) throws NameException, SystemException;
395 # [DDR Simple API] Service public PropertyValue getPropertyValue(Evidence evidence, String localPropertyName) throws NameException, SystemException;
396 # [DDR Simple API] Service public PropertyValue getPropertyValue(Evidence evidence, String localPropertyName, String localAspectName, String vocabularyIRI)
    throws NameException;
397 sub getPropertyValue {
398     my $this = shift;
399     my ($evidence, $p2, $p3, $p4) = @_;
400     my $localAspectName = undef;
401     my $localPropertyName = undef;
402     my $propertyRef = undef;
403     my $vocabIRI = undef;
404     if (ref($p2) eq 'PropertyName') {                                     # getPropertyValue(Evidence, PropertyName)
405         $localAspectName = $this->{REPOSITORY}->getDefaultAspect($p2->getLocalPropertyName(), $this->{DEFAULTVOCABULARY});
406         $localPropertyName = $p2->getLocalPropertyName();
407         $propertyRef = $this->newPropertyRef($p2, $localAspectName);
408         $vocabIRI = $propertyRef->getNamespace();
409     }
410     elsif (ref($p2) eq 'PropertyRef') {                                   # getPropertyValue(Evidence, PropertyRef)
411         $localAspectName = $p2->getAspectName();
412         $localPropertyName = $p2->getLocalPropertyName();
413         $propertyRef = $p2;
414         $vocabIRI = $propertyRef->getNamespace();
415     }
416     else {
417         if (!defined($p3)) {                                             # getPropertyValue(Evidence, String)
418             $vocabIRI = $this->{DEFAULTVOCABULARY};
419             $localAspectName = $this->{REPOSITORY}->getDefaultAspect($p2, $vocabIRI);
420         }
421         else {                                                           # getPropertyValue(Evidence, String, String, String)
422             $localAspectName = $p3;
423             $vocabIRI = $p4;
424         }
425         $localPropertyName = $p2;
```

```
426     $propertyRef = $this->newPropertyRef($this->newPropertyName($localPropertyName,$vocabIRI),$localAspectName);
427 }
428 my $actualPropertyValue = $this->{REPOSITORY}->getValue($evidence,$localPropertyName,$localAspectName,$vocabIRI);
429 my $pv = PropertyValue->new($propertyRef,$actualPropertyValue);
430 return $pv;
431 }
432
433 # [DDR Simple API] Service public PropertyName newPropertyName(String localPropertyName) throws NameException;
434 # [DDR Simple API] Service public PropertyName newPropertyName(String vocabularyIRI, String localPropertyName) throws NameException;
435 sub newPropertyName {
436     my $this = shift;
437     my $paramCount = @_;
438     my ($p1,$p2) = @_;
439     if ($paramCount == 1) {
440         return PropertyName->new($p1,$this->{DEFAULTVOCABULARY});
441     }
442     if ($paramCount == 2) {
443         return PropertyName->new($p1,$p2);
444     }
445     return undef;
446 }
447
448 # [DDR Simple API] Service public PropertyRef newPropertyRef()
449 # [DDR Simple API] Service public PropertyRef newPropertyRef(PropertyName propertyName)
450 # [DDR Simple API] Service public PropertyRef newPropertyRef(PropertyName propertyName, String localAspectName)
451 sub newPropertyRef {
452     my $this = shift;
453     my ($localPropertyName,$localAspectName) = @_;
454     if (defined $localPropertyName) {
455         if (defined $localAspectName) {
456             return PropertyRef->new($localPropertyName,$localAspectName);
457         }
458         else {
459             return PropertyRef->new($localPropertyName,$PropertyRef::NULL_ASPECT);
460         }
461     }
462     die "Unknown factory signature for newPropertyRef";
463 }
464
```

```
465 # [DDR Simple API] Service public Evidence newHTTPEvidence()
466 # [DDR Simple API] Service public Evidence newHTTPEvidence(Map map)    # Must pass 'map' by reference rather than by value
467 sub newHTTPEvidence {
468     my $this = shift;
469     my %hmap = %{(shift)};
470     if (%hmap) {
471         return HTTPEvidence->new(\%hmap);
472     }
473     else {
474         return HTTPEvidence->new();
475     }
476 }
477
478 # -----
479 package Evidence;
480
481 # Constructor is not part of the DDR Simple API specification
482 sub new {
483     my $pkg = shift;
484     my $name = shift;
485     my $evidence = {};
486     bless ($evidence, $pkg);
487     $evidence->{'map'} = ();
488     return $evidence;
489 }
490
491 # [DDR Simple API] Evidence public Boolean exists()
492 sub exists {
493     my ($evidence,$key) = @_;
494     return defined($evidence->{'map'}{$key});
495 }
496
497 # [DDR Simple API] Evidence public String get(String)
498 sub get {
499     my ($evidence,$key) = @_;
500     return $evidence->{'map'}{$key};
501 }
502
503 # [DDR Simple API] Evidence public void put(String key, String value)
```

```
504 sub put {
505     my ($evidence,$key,$val) = @_;
506     $evidence->{'map'}{$key} = $val;
507 }
508
509
510 # -----
511 package HTTPEvidence;    # This is NOT part of the DDR Simple API
512 BEGIN { @HTTPEvidence::ISA = qw( Evidence ); }
513
514 # public interface Evidence extends Map { }
515 # Constructor is not part of SimpleAPI specification
516 sub new {
517     my $pkg = shift;
518     my %initialhmap = %{(shift)};
519     my %hmap;
520     if (defined(%initialhmap)) {
521         %hmap = %initialhmap;
522     }
523     else {
524         %hmap = ();
525     }
526     my $evidence = {};
527     bless ($evidence, $pkg);
528     %{ $evidence->{'map'}} = %hmap;
529     return $evidence;
530 }
531
532 # -----
533 package PropertyName;
534
535 # Constructor is not part of SimpleAPI specification
536 sub new {
537     my $pkg = shift;
538     my $name = shift;
539     my $this = {};
540     bless ($this, $pkg);
541     $this->{LOCALNAME} = $name;
542     if (@_) {
```

```
543     my $namespace = shift;
544     $this->{NAMESPACE} = $namespace;
545 }
546 else {
547     $this->{NAMESPACE} = 'http://www.w3.org/2008/01/ddr-core-vocabulary';
548 }
549 return $this;
550 }
551
552 # [DDR Simple API] PropertyName public String getLocalPropertyName()
553 sub getLocalPropertyName {
554     my $this = shift;
555     return $this->{LOCALNAME};
556 }
557
558 # [DDR Simple API] PropertyName public String getNamespace()
559 # Returns the IRI of the vocabulary to which this named property belongs
560 sub getNamespace {
561     my $this = shift;
562     return $this->{NAMESPACE};
563 }
564
565 # Implementation-specific method
566 sub _hash {
567     my $this = shift;
568     return $this->{NAMESPACE} . ':' . $this->{LOCALNAME};
569 }
570
571 # -----
572 package PropertyRef;
573
574 # Constructor is not part of SimpleAPI specification
575 sub new {
576     my $pkg      = shift;
577     my $property = shift;
578     my $aspect   = shift;
579     my $this = bless {}, $pkg;
580     if (ref($property) eq 'PropertyName') {
581         $this->{PROPERTYNAME} = $property;
```

```
582     }
583     else {
584         $this->{PROPERTYNAME} = PropertyName->new($property);
585     }
586     # In an advanced version, aspect would be a special class
587     $this->{LOCALASPECTNAME} = $aspect;
588     return $this;
589 }
590
591 # [DDR Simple API] PropertyRef public String getLocalPropertyName()
592 sub getLocalPropertyName {
593     my $this = shift;
594     return $this->{PROPERTYNAME}->getLocalPropertyName();
595 }
596
597 # [DDR Simple API] PropertyRef public String getAspectName()
598 sub getAspectName {
599     my $this = shift;
600     return $this->{LOCALASPECTNAME};
601 }
602
603 # [DDR Simple API] PropertyRef public String getNamespace()
604 sub getNamespace {
605     my $this = shift;
606     my $namespace = $this->{PROPERTYNAME}->getNamespace();
607     if (!defined($namespace)) {
608         return $PropertyRef::NULL_ASPECT;
609     }
610     else {
611         return $namespace;
612     }
613 }
614
615 # [DDR Simple API]
616 $PropertyRef::NULL_ASPECT = '__NULL__';
617
618 # Implementation-specific method
619 sub _hash {
620     my $this = shift;
```

```
621     return $this->{PROPERTYNAME}->_hash() . '::' . $this->{LOCALASPECTNAME};
622 }
623
624 # -----
625 package PropertyValues;
626
627 # Constructor is not part of SimpleAPI specification
628 sub new {
629     my $pkg = shift;
630     my $name = shift;
631     my $this = {};
632     bless ($this, $pkg);
633     return $this;
634 }
635
636 # [DDR Simple API] PropertyValues public PropertyValue[] getAll()
637 sub getAll {
638     my $this = shift;
639     my @allPropertyValues = ();
640     foreach my $hash (sort keys %{$this}) { # sorting is an arbitrary implementation decision. Not necessary according to the spec.
641         my $propertyValue = $this->{$hash};
642         push(@allPropertyValues, $propertyValue);
643     }
644     return @allPropertyValues;
645 }
646
647 # [DDR Simple API] PropertyValues public PropertyValue getValue(PropertyRef prop) throws NameException
648 sub getValue {
649     my $this = shift;
650     my $pRef = shift; die('Got ' . ref($pRef) . ' when expecting PropertyRef') if ref($pRef) ne 'PropertyRef';
651     my $hash = $pRef->_hash();
652     my $value = $this->{$hash};
653     if (!defined $value) {
654         my $vocabIRI = $pRef->getNamespace();
655         my $aspectName = $pRef->getAspectName();
656         my $propname = $pRef->getLocalPropertyName();
657         die NameException->new($NameException::PROPERTY_NOT_RECOGNIZED, "NameException: $vocabIRI:$aspectName:$propname not found."
658     );
659     }
```

```
659     return $value;
660 }
661
662 # Not part of public interface specification
663 # Assumed to be implementation dependent and private
664 sub setValue {
665     my $this = shift;
666     my $pRef = shift; # assume to be a PropertyRef object
667     my $value = shift; # assume to be a PropertyValue object
668     $this->{$pRef->_hash()} = $value;
669 }
670
671 # -----
672 package PropertyValue;
673
674 # Constructor is not part of official specification
675 sub new {
676     my $pkg      = shift;
677     my $pRef      = shift;
678     my $actualValue = shift;
679     my $this = {};
680     bless ($this, $pkg);
681     $this->{PROPERTYREF} = $pRef;
682     $this->{VALUE} = $actualValue;
683     return $this;
684 }
685
686 # [DDR Simple API] PropertyValue public String getString() throws ValueException;
687 sub getString {
688     my $this = shift;
689     my $value = $this->{VALUE};
690     if (defined($value)) {
691         return '' . $value; # Always return some string, even if it's actually a representation of a non-string
692     }
693     else {
694         my $propName = $this->{PROPERTYREF}->getLocalPropertyName();
695         my $propAspect = $this->{PROPERTYREF}->getAspectName();
696         die ValueException->new($ValueException::NOT_KNOWN, "ValueException: no value exists for the $propAspect:$propName
property");

```

```
697     }
698 }
699
700 # [DDR Simple API] PropertyValue public boolean getBoolean() throws ValueException;
701 sub getBoolean {
702     my $this = shift;
703     my $value = $this->{VALUE};
704     if (defined($value)) {
705         return $value?1:0; # Perl doesn't have an internal Boolean representation, so use Perl magic
706     }
707     else {
708         my $propName = $this->{PROPERTYREF}->getLocalPropertyName();
709         my $propAspect = $this->{PROPERTYREF}->getAspectName();
710         die ValueException->new($ValueException::NOT_UNKNOWN, "ValueException: no value exists for the $propAspect:$propName
property");
711     }
712 }
713
714 # [DDR Simple API] PropertyValue public int getInteger() throws ValueException;
715 sub getInteger {
716     my $this = shift;
717     my $value = $this->{VALUE};
718     if (defined($value)) {
719         if ($value =~ /\d+$/) {
720             return $value;
721         }
722         else {
723             my $propName = $this->{PROPERTYREF}->getLocalPropertyName();
724             my $propAspect = $this->{PROPERTYREF}->getAspectName();
725             die ValueException->new($ValueException::INCOMPATIBLE_TYPES, "ValueException: the $propAspect:$propName property is not
an integer");
726         }
727     }
728     else {
729         my $propName = $this->{PROPERTYREF}->getLocalPropertyName();
730         my $propAspect = $this->{PROPERTYREF}->getAspectName();
731         die ValueException->new($ValueException::NOT_UNKNOWN, "ValueException: no value exists for the $propAspect:$propName
property");
732     }
}
```

```
733 }
734
735 # [DDR Simple API] PropertyValue public String[] getEnumeration() throws ValueException;
736 sub getEnumeration {
737     my $this = shift;
738     my $value = $this->{VALUE};
739     if (defined($value)) {
740         if (ref($value) eq 'ARRAY') {
741             return @{$value};
742         }
743         else {
744             return [ $value ]; # put single value into a single-cell array
745         }
746     }
747     else {
748         my $propName = $this->{PROPERTYREF}->getLocalPropertyName();
749         my $propAspect = $this->{PROPERTYREF}->getAspectName();
750         die ValueException->new($ValueException::NOT_KNOWN, "ValueException: no value exists for the $propAspect:$propName
property");
751     }
752 }
753 # [DDR Simple API] PropertyValue public float getFloat() throws ValueException;
754 # To Do
755 # [DDR Simple API] PropertyValue public double getDouble() throws ValueException;
756 # To Do
757 # [DDR Simple API] PropertyValue public long getLong() throws ValueException;
758 # To Do
759
760 # [DDR Simple API] PropertyValue public PropertyName getPropertyName();
761 sub getPropertyRef {
762     my $this = shift;
763     return $this->{PROPERTYREF};
764 }
765
766 # [DDR Simple API] PropertyValue public boolean exists();
767 sub exists {
768     my $this = shift;
769     return defined($this->{VALUE});
770 }
```

```
771
772 #####
773 ## Part 5 #
774 ## Exceptions thrown by various DDR Simple API methods. #
775 #####
776
777
778 # = = = = = EXCEPTION CLASSES = = = = =
779
780 # -----
781 # BaseException is implementation-specific. (Not part of DDR API)
782 package BaseException;
783
784 sub new {
785     my $pkg = shift;
786     my $message = shift;
787     my $self = bless { MESSAGE => $message }, $pkg;
788     return $self;
789 }
790
791 sub getMessage {
792     my $this = shift;
793     return $this->{MESSAGE};
794 }
795
796 # -----
797 package DDRException;
798 BEGIN { @DDRException::ISA = qw( BaseException ); }
799
800 use Carp;
801
802 # DDRException()
803 # DDRException(int code, String message)
804 # DDRException(int code, Throwable thr) // Throwable is not a Perl type
805 sub new {
806     my $pkg = shift;
807     my $self = bless { }, $pkg;
808     if (@_) {
809         my $code = shift;
```

```
810     $self->{CODE} = $code;
811     if (@_) {
812         my $message = shift;      # assume a message string. Throwable not supported.
813         $self->{MESSAGE} = $message;
814     }
815 }
816 ($cpkg, $cfile, $cline) = caller;
817 #print "$pkg created by $cpkg at line $cline. (" . $self{MESSAGE} . ")\n";
818 #confess();
819 return $self;
820 }
821
822 # [DDR Simple API] DDRException public int getCode()
823 sub getCode {
824     my $this = shift;
825     return $this->{CODE};
826 }
827
828 # [DDR Simple API] DDRException public String getMessage()
829 sub getMessage {
830     my $this = shift;
831     return $this->{MESSAGE};
832 }
833
834 # -----
835 package NameException;
836 BEGIN {
837     @NameException::ISA = qw( DDRException );
838     # [DDR Simple API]
839     $ASPECT_NOT_RECOGNIZED      = 800;
840     $PROPERTY_NOT_RECOGNIZED    = 100;
841     $VOCABULARY_NOT_RECOGNIZED = 200;
842 }
843
844 # -----
845 package SimpleException;
846 BEGIN { @SimpleException::ISA = qw( DDRException ); }
847
848 # -----
```

```
849 package ValueException;
850 BEGIN {
851     @ValueException::ISA = qw( DDRException );
852     # [DDR Simple API]
853     $INCOMPATIBLE_TYPES = 600;
854     $MULTIPLE_VALUES     = 1000;
855     $NOT_KNOWN           = 900;
856 }
857
858 # -----
859 package SystemException;
860 BEGIN {
861     @SystemException::ISA = qw( BaseException );
862     # [DDR Simple API]
863     $CANNOT_PROCEED = 500;
864     $INITIALIZATION = 400;
865 }
866
867 use Carp;
868
869 # SystemException()
870 # SystemException(int code, String message)
871 # SystemException(int code, Throwable thr)    // Throwable is not a Perl type
872 sub new {
873     my $pkg = shift;
874     my $self = bless { }, $pkg;
875     if (@_) {
876         my $code = shift;
877         $self->{CODE} = $code;
878         if (@_) {
879             my $message = shift;    # assume a message string. Throwable not supported.
880             $self->{MESSAGE} = $message;
881         }
882     }
883     ($cpkg, $cfile, $cline) = caller;
884     #print "$pkg created by $cpkg at line $cline. (" . $self{MESSAGE} . ")\n";
885     #confess();
886     return $self;
887 }
```

```
888
889 # [DDR Simple API] SystemException public int getCode()
890 sub getCode {
891     my $this = shift;
892     return $this->{CODE};
893 }
894
895 # [DDR Simple API] SystemException public String getMessage()
896 sub getMessage {
897     my $this = shift;
898     return $this->{MESSAGE};
899 }
900
901
902 #####
903 ## Part 6                                     #
904 ## A custom implementation of the back-end logic that retrieves actual data. #
905 ## Actual implementations of this part would probably interact with a database #
906 ## or expert system, or fuzzy logic or some other proprietary system.      #
907 ## This example hard-codes the data in-situ and does not connect elsewhere. #
908 ## Only a few of the DDR Core Vocabulary property terms are represented here. #
909 ## Context recognition depends solely on matching the User-Agent evidence.   #
910 ## Some pseudo-devices are hard-coded in this collection of device data.    #
911 ## Unknown/unavailable data are represented as 'undef'.                     #
912 #####
913
914
915
916 # CUSTOM REPOSITORY IMPLEMENTATION
917 # (Barely functional!)
918
919 # -----
920 package CustomDDRImplementation;
921
922 # Intentionally inefficient storage of device descriptions.
923 # Intentionally poor device recognition.
924 # If you want professional implementations, make or buy them.
925 sub new {
926     my $pkg = shift;
```

```

927 my $this = {};
928 bless ($this, $pkg);
929 $this->{DATAVERSION} = '080715';
930 $this->{VOCABULARIES} = ([ 'http://www.w3.org/2008/01/ddr-core-vocabulary' ]); # This repository only knows the Core Vocab
931 $this->{ASPECTS} = ([ 'device', 'webBrowser' ]); # All of the aspects known to the repository
932 $this->{USERAGENTS} = ([ # There are 4 devices known to this repository
933     'EI-emu (Gekoo; X11; watzit)',
934     'Mozthing1.2e (X11; en-US; v12)',
935     'Opella99 (Dash2; en-UK; mod-4; patched) nosuch/1255',
936     'FakeDevice'
937 ]);
938 $this->{'http://www.w3.org/2008/01/ddr-core-vocabulary'} = {
939     'vendor' => {
940         'device' => [ undef, undef, undef, 'Acme Co.' ], # only the vendor of FakeDevice is known
941         'webBrowser' => [ undef, undef, undef, undef ],
942         'DEFAULT' => 'device'
943     },
944     'model' => { # no information regarding models is known
945         'device' => [ undef, undef, undef, undef ],
946         'webBrowser' => [ undef, undef, undef, undef ],
947         'DEFAULT' => 'device'
948     },
949     'displayHeight' => { # some information about the display height is known
950         'device' => [ 260, 800, 140, undef ],
951         'webBrowser' => [ 240, 788, 136, undef ],
952         'DEFAULT' => 'device' # device takes precedence over webBrowser in the Core Vocab
953     },
954     'displayWidth' => {
955         'device' => [ 180, undef, 280, 120 ],
956         'webBrowser' => [ 160, 1012, 276, undef ],
957         'DEFAULT' => 'device' # device takes precedence over webBrowser in the Core Vocab
958     },
959     'imageFormatSupport' => {
960         'webBrowser' => [
961             [ 'gif87', 'gif89a', 'jpeg', 'png', ],
962             [ 'gif89a', 'jpeg', 'png', ],
963             [ 'gif87', 'gif89a', 'jpeg', ],
964             undef # repository does not know about the image support for FakeDevice
965         ],

```

```

966     'DEFAULT'      => 'webBrowser'
967 },
968     'cookieSupport' => {
969         'webBrowser' => [ undef, undef, undef, 1 ],
970         'DEFAULT'    => 'webBrowser'
971     },
972     'markupSupport' => {
973         'webBrowser' => [ undef, undef, undef, [ 'htmlBasic10', 'htmlMP10' ] ],
974         'DEFAULT'    => 'webBrowser'
975     }
976 };
977 return $this;
978 }
979
980 sub _getValueDirect {
981     my $this      = shift;
982     my $useragent  = shift; # just an ordinary string
983     my $vocabulary = shift; # just an ordinary string
984     my $property   = shift; # just an ordinary string
985     my $aspect     = shift; # just an ordinary string
986     my @agents = @{$this->{USERAGENTS}};
987     my $lastUAindex = $#agents;
988     my $i = 0;
989     while ($i <= $lastUAindex && $agents[$i] ne $useragent) {
990         $i++;
991     }
992     if ($i <= $lastUAindex) {
993         return $this->{$vocabulary}->{$property}->{$aspect}[$i];
994     }
995     return undef;
996 }
997
998 sub getValue {
999     my $this      = shift;
1000     my $evidence  = shift;
1001     my $localPropertyName = shift;
1002     my $localAspectName   = shift;
1003     my $namespace         = shift;
1004     my $ua = $evidence->get('User-Agent');

```

```
1005     return _getValueDirect($this,$ua,$namespace,$localPropertyName,$localAspectName);
1006 }
1007
1008 # returns all vocabularies (IRIs) supported by this custom implementation
1009 sub getVocabularies {
1010     my $this = shift;
1011     return $this->{VOCABULARIES};
1012 }
1013
1014 # returns all property names for the given vocabulary IRI
1015 sub getPropertyNames {
1016     my $this = shift;
1017     my $vocabulary = shift;
1018     my %v = %{$this->{$vocabulary}};
1019     my @result = sort keys %v;
1020     return \@result;
1021 }
1022
1023 # return all supported aspects for a given named property in a given vocabulary
1024 # (Note: this is implemented by scanning the available data)
1025 sub getSupportedAspects {
1026     my $this = shift;
1027     my $vocabulary = shift;
1028     my $propName = shift;
1029     my %repository = %{$this->{$vocabulary}};
1030     my %entries = %{$repository{$propName}};
1031     my @aspects = ();
1032     foreach my $a (keys %entries) {
1033         if ($a ne 'DEFAULT') { push(@aspects,$a); }
1034     }
1035     return \@aspects;
1036 }
1037
1038 # returns all aspects supported by this custom implementation
1039 # This is implemented by reading a previously stored summary from the repository
1040 sub getAllAspects {
1041     my $this = shift;
1042     return $this->{ASPECTS};
1043 }
```

```
1044
1045 # returns the default aspect for a named property in the given vocabulary
1046 sub getDefaultAspect {
1047     my $this      = shift;
1048     my $property   = shift;
1049     my $vocabulary = shift;
1050     return $this->{$vocabulary}->{$property}->{DEFAULT};
1051 }
1052
1053 # returns the version of the data that is available from the repository
1054 sub getDataVersion {
1055     my $this = shift;
1056     return $this->{DATAVERSION};
1057 }
1058
1059 #####
1060 # Part 7                                     #
1061 # The test methods derived from the published Java test class             #
1062 #####
1063
1064 # -----
1065 package DDRSimpleAPItest;
1066
1067 my $vocabularyIRI;
1068 my $aspect1;
1069 my $aspect2;
1070 my $totalPropertRefsInService;
1071
1072 my $localPropertyKnownString;
1073 my $localPropertyUnknownString;
1074 my $localAspectKnownString;
1075 my $localAspectUnknownString;
1076 my $localKnownStringValue;
1077
1078 my $localPropertyKnownInteger;
1079 my $localAspectKnownInteger;
1080 my $localKnownIntegerValue;
1081
1082 my $localPropertyKnownBoolean;
```

```
1083 my $localAspectKnownBoolean;
1084 my $localKnownBooleanValue;
1085
1086 my $localPropertyKnownEnumeration;
1087 my $localAspectKnownEnumeration;
1088 my @localKnownEnumerationValue;
1089
1090 my %report;
1091
1092 my $s;
1093 my $e;
1094
1095 sub initTest {
1096     $s = shift;
1097     $e = shift;
1098     $vocabularyIRI = shift;
1099     $aspect1 = shift;
1100     $aspect2 = shift;
1101     $totalPropertRefsInService = shift;
1102     $localPropertyKnownString = shift;
1103     $localPropertyUnknownString = shift;
1104     $localAspectKnownString = shift;
1105     $localAspectUnknownString = shift;
1106     $localKnownStringValue = shift;
1107     $localPropertyKnownInteger = shift;
1108     $localAspectKnownInteger = shift;
1109     $localKnownIntegerValue = shift;
1110     $localPropertyKnownBoolean = shift;
1111     $localAspectKnownBoolean = shift;
1112     $localKnownBooleanValue = shift;
1113     $localPropertyKnownEnumeration = shift;
1114     $localAspectKnownEnumeration = shift;
1115     @localKnownEnumerationValue = @_; # remaining parameters are the array of the test enumeration
1116     Clear();
1117 }
1118
1119 sub setReport {
1120     my $status = shift;
1121     my $testID = shift;
```

```
1122     $report{$testID} = $status?'Pass':'FAIL';
1123 }
1124
1125 sub Clear {
1126     my $testID = shift;
1127     if (defined($testID)) {
1128         $report{$testID} = '????';
1129     }
1130     else {
1131         %report = ();
1132     }
1133 }
1134
1135 sub getReport {
1136     Clear();
1137     Clear('#sec-Evidence');
1138     Clear('#sec-Evidence-get');
1139     Clear('#sec-Evidence-exists');
1140     Clear('#sec-Evidence-put');
1141
1142     Clear('#sec-PropertyName');
1143     Clear('#sec-PropertyName-getLocalPropertyName');
1144     Clear('#sec-PropertyName-getNamespace');
1145
1146     Clear('#sec-PropertyRef');
1147     Clear('#sec-PropertyRef-getLocalPropertyName');
1148     Clear('#sec-PropertyRef-getAspectName');
1149     Clear('#sec-PropertyRef-getNamespace');
1150
1151     Clear('#sec-PropertyValue');
1152     Clear('#sec-PropertyValue-getXXX Double');      # Untested
1153     Clear('#sec-PropertyValue-getXXX Long');        # Untested
1154     Clear('#sec-PropertyValue-getXXX String');
1155     Clear('#sec-PropertyValue-getXXX Boolean');
1156     Clear('#sec-PropertyValue-getXXX Integer');
1157     Clear('#sec-PropertyValue-getXXX Enumeration');
1158     Clear('#sec-PropertyValue-getXXX Float');      # Untested
1159     Clear('#sec-PropertyValue-exists');
1160     Clear('#sec-PropertyValue-getPropertyRef');
```

```
1161
1162     Clear('#sec-PropertyValues');
1163     Clear('#sec-PropertyValues-getAll');
1164     Clear('#sec-PropertyValues-getValue');
1165
1166     Clear('#sec-Service');
1167     Clear('#sec-Service-newHTTPEvidence-1');
1168     Clear('#sec-Service-newHTTPEvidence-2');
1169     Clear('#sec-Service-newPropertyName-1');
1170     Clear('#sec-Service-newPropertyName-2');
1171     Clear('#sec-Service-newPropertyRef-1');
1172     Clear('#sec-Service-newPropertyRef-2');
1173     Clear('#sec-Service-newPropertyRef-3');
1174     Clear('#sec-Service-getPropertyValues-1');
1175     Clear('#sec-Service-getPropertyValues-2');
1176     Clear('#sec-Service-getPropertyValues-3');
1177     Clear('#sec-Service-getPropertyValues-4');
1178     Clear('#sec-Service-getPropertyValue-1');
1179     Clear('#sec-Service-getPropertyValue-2');
1180     Clear('#sec-Service-getPropertyValue-3');
1181     Clear('#sec-Service-getPropertyValue-4');
1182     Clear('#sec-Service-getImplementationVersion');
1183     Clear('#sec-Service-getDataVersion');
1184     Clear('#sec-Service-listPropertyRefs');
1185     Clear('#sec-Service-initialize');
1186
1187     eval { # TRY
1188         local $SIG{'__DIE__'};
1189
1190         # Service tests included in other tests
1191         my $factoryCreatedEvidence = 0; #false
1192         my $factoryCreatedPropertyName = 0; #false
1193         my $factoryCreatedPropertyRef = 0; #false
1194         my $obtainedPropertyValueInstance = 0; #false
1195         my $obtainedPropertyValuesInstance = 0; #false
1196
1197         # Evidence
1198         {
1199             my $putCausedNoException = 0; #false
```

```

1200 my $getReturnedCorrectValue = 0; #false
1201 my $getDoesNotKnowInputs = 0; #false
1202 my $existsWorks = 0; #false
1203
1204 my $evidence = $s->newHTTPEvidence();
1205 $factoryCreatedEvidence = (defined($evidence) && $evidence->isa(Evidence));
1206 $evidence->put('TestHeader', 'TestHeaderValue');
1207 $putCausedNoException = 1; #true
1208 $getReturnedCorrectValue = $evidence->get('TestHeader') eq 'TestHeaderValue';
1209 $existsWorks = $evidence->exists('TestHeader') && !$evidence->exists('UnknownHeader');
1210 eval {
1211     $getDoesNotKnowInputs = (!defined($evidence->get('UnknownHeader')) || '' eq $evidence->get('UnknownHeader'));
1212 };
1213 if ($@) {
1214     $getDoesNotKnowInputs = 1; #true # Throwing an exception is also a valid response to not finding the header
1215 }
1216
1217 setReport($factoryCreatedEvidence, '#sec-Service-newHTTPEvidence-1');
1218 setReport($putCausedNoException, '#sec-Evidence-put'); # Only need to be sure put() didn't cause an exception
1219 setReport($getReturnedCorrectValue && $getDoesNotKnowInputs, '#sec-Evidence-get');
1220 setReport($existsWorks, '#sec-Evidence-exists');
1221 setReport($putCausedNoException && $getReturnedCorrectValue && $getDoesNotKnowInputs && $existsWorks, '#sec-Evidence');
1222 }
1223
1224 # PropertyName
1225 {
1226     my $knowsName = 0; #false
1227     my $knowsNamespace = 0; #false
1228
1229     my $propertyName = $s->newPropertyName($localPropertyKnownString, $vocabularyIRI);
1230     $factoryCreatedPropertyName = 1; #true
1231     $knowsName = $localPropertyKnownString eq $propertyName->getLocalPropertyName();
1232     $knowsNamespace = $vocabularyIRI eq $propertyName->getNamespace();
1233
1234     setReport($factoryCreatedPropertyName, '#sec-Service-newPropertyName-2');
1235     setReport($knowsName, '#sec-PropertyName-getLocalPropertyName');
1236     setReport($knowsNamespace, '#sec-PropertyName-getNamespace');
1237     setReport($knowsName && $knowsNamespace, '#sec-PropertyName');
1238 }

```

```
1239
1240     # PropertyRef
1241     {
1242         my $knowsName = 0; #false
1243         my $knowsNamespace = 0; #false
1244         my $knowsAspect = 0; #false
1245
1246         my $propertyRef = $s->newPropertyRef($s->newPropertyName($localPropertyKnownString, $vocabularyIRI),
$localAspectKnownString);
1247         $factoryCreatedPropertyRef = defined($propertyRef) && $propertyRef->isa(PropertyRef);
1248         $knowsName = $localPropertyKnownString eq $propertyRef->getLocalPropertyName();
1249         $knowsAspect = $localAspectKnownString eq $propertyRef->getAspectName();
1250         $knowsNamespace = $vocabularyIRI eq $propertyRef->getNamespace();
1251
1252         setReport($factoryCreatedPropertyRef, '#sec-Service-newPropertyRef-3');
1253         setReport($knowsName, '#sec-PropertyRef-getLocalPropertyName');
1254         setReport($knowsAspect, '#sec-PropertyRef-getAspectName');
1255         setReport($knowsNamespace, '#sec-PropertyRef-getNamespace');
1256         setReport($knowsName && $knowsAspect && $knowsNamespace, '#sec-PropertyRef');
1257     }
1258
1259     # PropertyValue
1260     {
1261         my $obtainedExistingValue = 0; #false
1262         my $detectedUnknownValue = 0; #false
1263         my $gotSomeStringRepresentation = 0; #false
1264         my $gotPropertyRef = 0; #false
1265         my $gotCorrectString = 0; #false
1266         my $gotCorrectInteger = 0; #false
1267         my $gotCorrectBoolean = 0; #false
1268         my $gotCorrectEnumeration = 0; #false
1269
1270         my $propertyValue = $s->getPropertyValue($e, $localPropertyKnownString, $localAspectKnownString, $vocabularyIRI);
1271         $obtainedPropertyValueInstance = defined($propertyValue) && $propertyValue->isa(PropertyValue);
1272         $obtainedExistingValue = $propertyValue->exists();
1273         eval { # TRY
1274             my $propertyValueBad = $s->getPropertyValue($e, $localPropertyUnknownString, $localAspectUnknownString,
$vocabularyIRI);
1275         };
1276     }
```

```

1276     if ($@) {
1277         if ($@->isa(NameException)) {
1278             $detectedUnknownValue = 1; #true # Implementations are free to use alternative codes, so just check for the exception
1279         }
1280         else {
1281             die $@;
1282         }
1283     }
1284     my $stringRepresentation = $propertyValue->getString();
1285     $gotSomeStringRepresentation = defined($stringRepresentation) && $stringRepresentation ne '';
1286     $gotCorrectString = defined($stringRepresentation) && $stringRepresentation eq $localKnownStringValue;
1287     my $propertyRef = $propertyValue->getPropertyRef();
1288     $gotPropertyRef = (
1289         defined($propertyRef) && $propertyRef->isa(PropertyRef) &&
1290         $localPropertyKnownString eq $propertyRef->getLocalPropertyName() &&
1291         $localAspectKnownString eq $propertyRef->getAspectName() &&
1292         $vocabularyIRI eq $propertyRef->getNamespace()
1293     );
1294     my $propertyValueInteger = $s->getPropertyValue($e, $localPropertyKnownInteger, $localAspectKnownInteger,
1295 $vocabularyIRI);
1296     $gotCorrectInteger = ($propertyValueInteger->getInteger() == $localKnownIntegerValue);
1297     my $propertyValueBoolean = $s->getPropertyValue($e, $localPropertyKnownBoolean, $localAspectKnownBoolean,
1298 $vocabularyIRI);
1299     $gotCorrectBoolean = ($propertyValueBoolean->getBoolean() == $localKnownBooleanValue);
1300     my $propertyValueEnumeration = $s->getPropertyValue($e, $localPropertyKnownEnumeration, $localAspectKnownEnumeration,
1301 $vocabularyIRI);
1302     my @enumeratedValue = $propertyValueEnumeration->getEnumeration();
1303     $gotCorrectEnumeration = (scalar(@enumeratedValue) == scalar(@localKnownEnumerationValue));
1304     if ($gotCorrectEnumeration) {
1305         for (my $i = 0; $i < scalar(@enumeratedValue); $i++) {
1306             # Warning: the order of values in the enumeration is prescribed for this test,
1307             # although nothing has been said regarding the significance of any such ordering.
1308             $gotCorrectEnumeration = $gotCorrectEnumeration && $localKnownEnumerationValue[$i] eq $enumeratedValue[$i];
1309         }
1310     }
1311     # not tested: #sec-PropertyValue-getXXX Double
1312     # not tested: #sec-PropertyValue-getXXX Long
1313     # not tested: #sec-PropertyValue-getXXX Float

```

```

1312     setReport($obtainedExistingValue && $gotCorrectString && $gotSomeStringRepresentation, '#sec-PropertyValue-getXXX
String');
1313     setReport($gotCorrectInteger, '#sec-PropertyValue-getXXX Integer');
1314     setReport($gotCorrectBoolean, '#sec-PropertyValue-getXXX Boolean');
1315     setReport($gotCorrectEnumeration, '#sec-PropertyValue-getXXX Enumeration');
1316     setReport($obtainedExistingValue && detectedUnknownValue, '#sec-PropertyValue-exists');
1317     setReport($gotPropertyRef, '#sec-PropertyValue-getPropertyRef');
1318     setReport($gotCorrectInteger && $gotCorrectBoolean && $gotCorrectEnumeration && $gotPropertyRef, '#sec-PropertyValue');
1319     setReport($obtainedPropertyValueInstance, '#sec-Service-getPropertyValue-4');
1320 }
1321
1322 # PropertyValue
1323 {
1324     my $gotCorrectArray = 0; #false
1325     my $gotCorrectValue = 0; #false
1326
1327     my @propertyRefs = [
1328         $s->newPropertyRef($s->newPropertyName($localPropertyKnownString, $vocabularyIRI), $localAspectKnownString),
1329         $s->newPropertyRef($s->newPropertyName($localPropertyKnownInteger, $vocabularyIRI), $localAspectKnownInteger),
1330         $s->newPropertyRef($s->newPropertyName($localPropertyKnownBoolean, $vocabularyIRI), $localAspectKnownBoolean)
1331     ];
1332     my $propertyValues = $s->getPropertyValues($e, @propertyRefs);
1333     $obtainedPropertyValuesInstance = (defined($propertyValues) && $propertyValues->isa(PropertyValues));
1334     my @propertyValueArray = $propertyValues->getAll();
1335     my $retrievedString = undef;
1336     my $retrievedInteger = undef;
1337     my $retrievedBoolean = undef;
1338     for (my $i = 0; $i < 3; $i++) {
1339         my $lpn = $propertyValueArray[$i]->getPropertyRef()->getLocalPropertyName();
1340         if ($lpn eq 'vendor') { $retrievedString = $propertyValueArray[$i]->getString(); }
1341         elsif ($lpn eq 'displayWidth') { $retrievedInteger = $propertyValueArray[$i]->getInteger(); }
1342         elsif ($lpn eq 'cookieSupport') { $retrievedBoolean = $propertyValueArray[$i]->getBoolean(); }
1343     }
1344     $gotCorrectArray = (
1345         defined(@propertyValueArray) && scalar(@propertyValueArray) == 3 &&
1346         $localKnownStringValue eq $retrievedString &&
1347         $localKnownIntegerValue == $retrievedInteger &&
1348         $localKnownBooleanValue == $retrievedBoolean
1349     );

```

```

1350 my $innerPropertyRef = $s->newPropertyRef(
1351     $s->newPropertyName($localPropertyKnownInteger, $vocabularyIRI), $localAspectKnownInteger
1352 );
1353 my $innerPropertyValue = $propertyValues->getValue($innerPropertyRef);
1354 $gotCorrectValue = ($innerPropertyValue->getInteger() == $localKnownIntegerValue);
1355
1356 setReport($gotCorrectArray, '#sec-PropertyValues-getAll');
1357 setReport($gotCorrectValue, '#sec-PropertyValues-getValue');
1358 setReport($gotCorrectArray && $gotCorrectValue, '#sec-PropertyValues');
1359 setReport($obtainedPropertyValuesInstance, '#sec-Service-getPropertyValues-4');
1360 }
1361
1362 # Service
1363 # Tests cover features not already covered by previous tests
1364 {
1365     my $gotListOfProperties = 0; #false
1366     my $evidenceViaMapOK = 0; #false
1367     my $propNameDefaultIRIOK = 0; #false
1368     my $propRefStringOK = 0; #false
1369     my $propRefPropNameOK = 0; #false
1370     my $propValuesEvidenceOK = 0; #false
1371     my $propValuesEvidenceAspectOK = 0; #false
1372     my $propValuesEvidenceAspectVocabOK = 0; #false
1373     my $propValueEvidencePropRefOK = 0; #false
1374     my $propValueEvidencePropNameOK = 0; #false
1375     my $propValueEvidenceNameOK = 0; #false
1376
1377     my @propertyRefArray = @{$s->listPropertyRefs()};
1378     if (defined(@propertyRefArray) && scalar(@propertyRefArray) == $totalPropertRefsInService) {
1379         $gotListOfProperties = 1; #true
1380     }
1381
1382     # #sec-Service-newHTTPEvidence-2
1383     my %hmap = ();
1384     $hmap{'X-Header1'} = 'HeaderValue1';
1385     $hmap{'X-Header2'} = 'HeaderValue2';
1386     $hmap{'X-Header1'} = 'HeaderValue3';
1387     my $e2 = $s->newHTTPEvidence(\%hmap);
1388     if ('HeaderValue3' eq $e2->get('X-Header1') && 'HeaderValue2' eq $e2->get('X-Header2')) {

```

```
1389     $evidenceViaMapOK = $e2->exists('X-Header1') && $e2->exists('X-Header2') && !$e2->exists('X-Header3');
1390 }
1391
1392 my $pn = $s->newPropertyName($localPropertyKnownString, $vocabularyIRI);
1393 my $pr = $s->newPropertyRef($pn, $localAspectKnownString);
1394
1395 # #sec-Service-newPropertyName-1
1396 # Test assumes that default vocabulary was used in the class constructor.
1397 my $pn1 = $s->newPropertyName($localPropertyKnownString);
1398 $propNameDefaultIRIOK = $pn1->getLocalPropertyName() eq $localPropertyKnownString && $pn1->getNamespace() eq
    $vocabularyIRI;
1399
1400 # #sec-Service-newPropertyRef-1
1401 # Test assumes that default vocabulary was used in the class constructor.
1402 my $pr1 = $s->newPropertyRef($localPropertyKnownString);
1403 $propRefStringOK = $pr1->getLocalPropertyName() eq $localPropertyKnownString &&
    $pr1->getNamespace() eq $vocabularyIRI && $pr1->getAspectName() eq $PropertyRef::NULL_ASPECT;
1404
1405 # #sec-Service-newPropertyRef-2
1406 my $pr2 = $s->newPropertyRef($pn1);
1407 $propRefPropNameOK = $pr2->getLocalPropertyName() eq $localPropertyKnownString &&
    $pr2->getNamespace() eq $vocabularyIRI && $pr2->getAspectName() eq $PropertyRef::NULL_ASPECT;
1408
1409 # #sec-Service-getPropertyValues-1
1410 my $pvs1 = $s->getPropertyValues($e);
1411 $propValuesEvidenceOK = $pvs1->getValue($pr)->getString() eq $localKnownStringValue;
1412
1413 # #sec-Service-getPropertyValues-2
1414 my $pvs2 = $s->getPropertyValues($e, $localAspectKnownString);
1415 $propValuesEvidenceAspectOK = $pvs2->getValue($pr)->getString() eq $localKnownStringValue;
1416
1417 # #sec-Service-getPropertyValues-3
1418 my $pvs3 = $s->getPropertyValues($e, $localAspectKnownString, $vocabularyIRI);
1419 $propValuesEvidenceAspectVocabOK = $pvs3->getValue($pr)->getString() eq $localKnownStringValue;
1420
1421 # #sec-Service-getPropertyValue-1
1422 my $pv1 = $s->getPropertyValue($e, $pr);
1423 $propValueEvidencePropRefOK = $pv1->getString() eq $localKnownStringValue;
1424
1425
1426
```

```

1427     # #sec-Service-getPropertyValue-2
1428     my $pv2 = $s->getPropertyValue($e, $pn);
1429     $propValueEvidencePropNameOK = $pv2->getString() eq $localKnownStringValue;
1430
1431     # #sec-Service-getPropertyValue-3
1432     my $pv3 = $s->getPropertyValue($e, $localPropertyKnownString);
1433     $propValueEvidenceNameOK = $pv3->getString() eq $localKnownStringValue;
1434
1435     setReport($evidenceViaMapOK, '#sec-Service-newHTTPEvidence-2');
1436     setReport($propNameDefaultIRIOK, '#sec-Service-newPropertyName-1');
1437     setReport($propRefStringOK, '#sec-Service-newPropertyRef-1');
1438     setReport($propRefPropNameOK, '#sec-Service-newPropertyRef-2');
1439     setReport($propValuesEvidenceOK, '#sec-Service-getPropertyValues-1');
1440     setReport($propValuesEvidenceAspectOK, '#sec-Service-getPropertyValues-2');
1441     setReport($propValuesEvidenceAspectVocabOK, '#sec-Service-getPropertyValues-3');
1442     setReport($propValueEvidencePropRefOK, '#sec-Service-getPropertyValue-1');
1443     setReport($propValueEvidencePropNameOK, '#sec-Service-getPropertyValue-2');
1444     setReport($propValueEvidenceNameOK, '#sec-Service-getPropertyValue-3');
1445     setReport(defined($s->getImplementationVersion()), '#sec-Service-getImplementationVersion'); # Just has to work without
causing an exception
1446     setReport(defined($s->getDataVersion()), '#sec-Service-getDataVersion'); # Just has to work without causing an exception
1447     setReport($gotListOfProperties, '#sec-Service-listPropertyRefs');
1448     setReport(1, '#sec-Service-initialize'); # 1=true Just has to work without causing an exception
1449     setReport(
1450         $factoryCreatedEvidence && $factoryCreatedPropertyName && $factoryCreatedPropertyRef &&
1451         $obtainedPropertyValueInstance && $obtainedPropertyValuesInstance &&
1452         $evidenceViaMapOK && $propNameDefaultIRIOK && $propRefStringOK && $propRefPropNameOK &&
1453         $propValuesEvidenceOK && $propValuesEvidenceAspectOK && $propValuesEvidenceAspectVocabOK &&
1454         $propValueEvidencePropRefOK && $propValueEvidencePropNameOK &&
1455         $propValueEvidenceNameOK && $gotListOfProperties &&
1456         defined($s->getImplementationVersion()) && defined($s->getDataVersion())
1457         , '#sec-Service');
1458     }
1459
1460 };
1461 if ($@) { # exception trap
1462     if (!$@->isa('BaseException')) {
1463         print "Unknown exception: $@\n";
1464     }

```

```
1465     die $@;
1466 }
1467 return %report;
1468 }
1469
1470 __END__;
1471
1472 #####
1473 ## Output #
1474 ## When executed, the program generates the following output (except for the 0x... memory ref): #
1475 ## #
1476 ## Using API (1.1.0 http://www.w3.org/TR/2008/WD-DDR-Simple-API-20080404/) with Repository (080715) #
1477 ## Evidence from the delivery context: #
1478 ##   User-Agent: Mozthing1.2e (X11; en-US; v12) #
1479 ##   Accept:text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */* #
1480 ## Getting specific software values for this context: #
1481 ##   Browser Display Height : 788 #
1482 ##   Browser Display Width  : 1012 #
1483 ## Getting specific hardware values for this context: #
1484 ##   Physical Display Height : 800 #
1485 ## Getting all values for this context: #
1486 ##   displayHeight is 800 #
1487 ##   imageFormatSupport is ARRAY(0x1ca8a0c) #
1488 ## Getting image format support for this context: #
1489 ## Getting information via Simple API methods only: #
1490 ##   Device Height = 800 #
1491 ## Getting information via implementation specific constructor: #
1492 ##   Device Height = 800 #
1493 ## Getting information via PropertyRef: #
1494 ##   Browser height = 788 #
1495 ## Getting information via custom convenience methods: #
1496 ##   Device Height = 800 #
1497 ##   Device Width  = 000000 #
1498 ##   Image formats = gif89a,jpeg,png #
1499 ## BEGIN Standard API tests ----- #
1500 ##   #sec-Evidence = Pass #
1501 ##   #sec-Evidence-exists = Pass #
1502 ##   #sec-Evidence-get = Pass #
1503 ##   #sec-Evidence-put = Pass #
```

1504	##	#sec-PropertyName = Pass	#
1505	##	#sec-PropertyName-getLocalPropertyName = Pass	#
1506	##	#sec-PropertyName-getNamespace = Pass	#
1507	##	#sec-PropertyRef = Pass	#
1508	##	#sec-PropertyRef-getAspectName = Pass	#
1509	##	#sec-PropertyRef-getLocalPropertyName = Pass	#
1510	##	#sec-PropertyRef-getNamespace = Pass	#
1511	##	#sec-PropertyValue = Pass	#
1512	##	#sec-PropertyValue-exists = Pass	#
1513	##	#sec-PropertyValue-getPropertyRef = Pass	#
1514	##	#sec-PropertyValue-getXXX Boolean = Pass	#
1515	##	#sec-PropertyValue-getXXX Double = ????	#
1516	##	#sec-PropertyValue-getXXX Enumeration = Pass	#
1517	##	#sec-PropertyValue-getXXX Float = ????	#
1518	##	#sec-PropertyValue-getXXX Integer = Pass	#
1519	##	#sec-PropertyValue-getXXX Long = ????	#
1520	##	#sec-PropertyValue-getXXX String = Pass	#
1521	##	#sec-PropertyValues = Pass	#
1522	##	#sec-PropertyValues-getAll = Pass	#
1523	##	#sec-PropertyValues-getValue = Pass	#
1524	##	#sec-Service = Pass	#
1525	##	#sec-Service-getDataVersion = Pass	#
1526	##	#sec-Service-getImplementationVersion = Pass	#
1527	##	#sec-Service-getPropertyValue-1 = Pass	#
1528	##	#sec-Service-getPropertyValue-2 = Pass	#
1529	##	#sec-Service-getPropertyValue-3 = Pass	#
1530	##	#sec-Service-getPropertyValue-4 = Pass	#
1531	##	#sec-Service-getPropertyValues-1 = Pass	#
1532	##	#sec-Service-getPropertyValues-2 = Pass	#
1533	##	#sec-Service-getPropertyValues-3 = Pass	#
1534	##	#sec-Service-getPropertyValues-4 = Pass	#
1535	##	#sec-Service-initialize = Pass	#
1536	##	#sec-Service-listPropertyRefs = Pass	#
1537	##	#sec-Service-newHTTPEvidence-1 = Pass	#
1538	##	#sec-Service-newHTTPEvidence-2 = Pass	#
1539	##	#sec-Service-newPropertyName-1 = Pass	#
1540	##	#sec-Service-newPropertyName-2 = Pass	#
1541	##	#sec-Service-newPropertyRef-1 = Pass	#
1542	##	#sec-Service-newPropertyRef-2 = Pass	#

```
1543  ##      #sec-Service-newPropertyRef-3 = Pass                                     #
1544  ## END      Standard API tests -----#
1545  ###
1546  #####
1547
1548
1549
```