Open Geospatial Consortium, Inc.

Date: 2009-08-05

Reference number of this document: OGC 09-006

Version: 0.3.0

Category: Public Engineering Report

Editor: Keith Pomakis

OGC® OWS-6 DSS Engineering Report - SOAP/XML and REST in WMTS

Document type:	OpenGIS® Public Engineering Report
Document subtype:	NA
Document stage:	Approved for Public Release

Preface

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by OpenOffice, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Forward

OWS-6 Testbed

OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver Engineering Reports and Chnage Requests into the OGC Specification Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

In April 2008, the OGC issued a call for sponsors for an OGC Web Services, Phase 6 (OWS-6) Testbed activity. The activity completed in June 2009. There is a series of on-line demonstrations available here: http://www.opengeospatial.org/pub/www/ows6/index.html
The OWS-6 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommended to the sponsors that the content of the OWS-6 initiative be organized around the following threads:

1. Sensor Web Enablement (SWE)
2. Geo Processing Workflow (GPW)
3. Aeronautical Information Management (AIM)
4. Decision Support Services (DSS)
5. Compliance Testing (CITE)

The OWS-6 sponsoring organizations were:

- U.S. National Geospatial-Intelligence Agency (NGA)
- Joint Program Executive Office for Chemical and Biological Defense (JPEO-CBD)
- GeoConnections - Natural Resources Canada
- U.S. Federal Aviation Agency (FAA)
- EUROCONTROL
- EADS Defence and Communications Systems
- US Geological Survey
- Lockheed Martin
- BAE Systems

- ERDAS, Inc.

The OWS-6 participating organizations were:
52North, AM Consult, Carbon Project, Charles Roswell, Compusult, con terra, CubeWerx, ESRI, FedEx, Galdos, Geomatys, GIS.FCU, Taiwan, GMU CSISS, Hitachi Ltd., Hitachi Advanced Systems Corp, Hitachi Software Engineering Co., Ltd., iGSI, GmbH, interactive instruments, lat/lon, GmbH, LISAsoft, Luciad, Lufthansa, NOAA MDL, Northrop Grumman TASC, OSS Nokalva, PCAvionics, Snowflake, Spot Image/ESA/Spacebel, STFC, UK, UAB CREAF, Univ Bonn Karto, Univ Bonn IGG, Univ Bunderswehr, Univ Muenster IfGI, Vightel, Yumetech.

Contents

Page

Listings..Page

OGC® OWS-6 DSS Engineering Report - SOAP/XML and REST in WMTS

1 Introduction

1.1 Scope

This OGC® document reports the results achieved in the Decision Support Services (DSS) subtask of the OWS-6 testbed initiative as it relates to the development of SOAP/XML and REST interfaces for the Web Map Tiling Service (WMTS).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization	E-Mail Address
Keith Pomakis	CubeWerx Inc.	pomakis@cubewerx.com
Pat Cappelaere	Vightel Corporation	pat@cappelaere.com
Jim Groffen	LISAsoft	jim.groffen@lisasoft.com
Alessandro Triglia	OSS Nokalva, Inc.	sandro@oss.com

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2008-12-12	0.0.1	K. Pomakis		First draft
2009-01-13	0.0.2	K. Pomakis		Second draft
2009-02-02	0.0.3	K. Pomakis		Third draft
2009-04-13	1.0.0	K. Pomakis		OWS-6 DSS deliverable
08-03-09	0.3.0	Carl Reed	Various	Prepare for publication

1.4 Future work

See Clause 8 for a discussion on recommended future work.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 05-009r3 (September 2007), *Template for OWS interface specifications*, Arliss Whiteside, ed.

OGC 05-057r6 (to be published), *OpenGIS® Web Map Tiling Service Implementation Standard*, Joan Masó, Keith Pomakis and Núria Julià, eds.

OGC 06-094r1 (February 2008), *OWS Common change request: Add SOAP encoding*, Arliss Whiteside and Bastian Schäffer, eds.

OGC 06-121r3 (February 2007), *OpenGIS® Web Services Common Specification, version 1.1.0 with Corrigendum 1*, Arliss Whiteside, ed.

OGC 08-009r1 (January 2008), *OWS 5 SOAP/WSDL Common Engineering Report*, Bastian Schäffer, ed.

RFC 2616 (June 1999), *Hypertext Transfer Protocol – HTTP/1.1*, R. Fielding *et al.*, eds., http://www.w3.org/Protocols/rfc2616/rfc2616.html

NOTE The *OpenGIS® Web Services Common Specification* [OGC 06-121r3] contains a list of normative references that are also relevant to this engineering report.

3 Conventions

3.1 Abbreviated terms

JSON	JavaScript Object Notation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
WSDL	Web Service Description Language

4 Engineering Report overview

This OGC™ document reports the results achieved in the Decision Support Services (DSS) subtask of the OWS-6 testbed initiative as it relates to the development of SOAP/XML and REST interfaces for the Web Map Tiling Service (WMTS).

The various interfaces presented in this engineering report are related only in their association as alternative encodings of the same operation requests. It is possible in theory to utilize two or more of these interfaces together in the same request chain by using a special proxy service to translate from one encoding to another. However, this engineering report makes no attempt to formalize such a proxy service.

Following the guidance of the *Template for OWS interface standards* [OGC 05-009r3] and the *OWS 5 SOAP/WSDL Common Engineering Report* [OGC 08-009r1], the WMTS SOAP/XML interface that this enginerring report recommends is relatively clear and uncontroversial.

SOAP relies on XML as its message syntax, and the *document-literal* style of SOAP (which is used in OGC standards) requires that all the request data or response data of each operation be encoded as a single XML element and be carried in the body of a SOAP message. Therefore, while a “direct” XML interface (one in which all the request or response data of each operation is encoded as a single XML element) has not been stated as a requirement for the WMTS specification, such an interface is a natural companion to the SOAP interface. For this reason, this engineering report also introduces an XML interface for the WMTS.

The REST interface, and indeed the whole resource-oriented approach, that this discussion paper recommends for WMTS is likely to be controversial since it is one of the first REST interfaces attempted by an OGC standard. Therefore, there is very little formal guidance that has community agreement. Furthermore, OGC standards have traditionally been defined with procedural-style interfaces, and the WMTS specification is no exception. Attempting to maintain these procedural-style interfaces while grafting in what the majority of the community would consider an appropriate REST interface, and to do so in a non-convoluted way, is no easy task.

5 WMTS XML interface

5.1 Declaration of support

A WMTS server that supports XML encodings may declare such support support by means of the OperationsMetadata section of its Capabilities document. As per Clause 8.3.2 of [OGC 05-009r3], this declaration is accomplished by specifying “XML” as an allowed value of the "PostEncoding" constraint within the DCP/HTTP/Post element of each operation that has support for the XML encoding. An example is presented in Listing 1:

```
<Operation name="GetTile">
   <DCP>
      <HTTP>
         <Post xlink:href="wmtsBaseUrl">
            <Constraint name="PostEncoding">
               <AllowedValues>
                  …
                  <Value>XML</Value>
                  …
               </AllowedValues>
            </Constraint>
         </Post>
      </HTTP>
   </DCP>
</Operation>
```

Listing 1: Example declaration of support for XML interface

If the XML encoding of an operation has a different base URL (a.k.a. "connect point") from the other encoding(s) of the operation, then the Post element may be repeated with the appropriate constraint(s) declared for each one.

Support for XML encodings is optional in WMTS (even in the presence of support for SOAP encodings).

5.2 Operation encodings

5.2.1 GetCapabilities

5.2.1.1 Request

The XML encoding of the GetCapabilities operation request shall be a GetCapabilities element conforming to the following XML schema:

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
      xmlns:ows=”http://www.opengis.net/ows/1.1”
      targetNamespace=”http://www.opengis.net/wmts/1.0”
      elementFormDefault=”qualified”
      attributeFormDefault=”unqualified”>
   <element name="GetCapabilities">
      <complexType>
         <complexContent>
            <extension base="ows:GetCapabilitiesType">
               <attribute name="service" type="ows:ServiceType"
                     use="required" fixed="WMTS"/>
            </extension>
         </complexContent>
      </complexType>
   </element>
</schema>
```

Listing 2: Schema of XML-encoded GetCapabilities request

An example is presented in Listing 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetCapabilities service="WMTS" xmlns="http://www.opengis.net/wmts/1.0">
   <AcceptVersions>
      <Version>1.0.0</Version>
   </AcceptVersions>
   <AcceptFormats>
      <OutputFormat>text/xml</OutputFormat>
   </AcceptFormats>
</GetCapabilities>
```

Listing 3: Example XML-encoded GetCapabilities request

5.2.1.2 Successful response

The response of a successful XML-encoded GetCapabilities operation request shall be a Capabilities document as defined in Clause 7.2.3 of [OGC 07-057r6].

5.2.1.3 Exception response

The response of an unsuccessful XML-encoded GetCapabilities operation request shall be an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3].

5.2.2 GetTile

5.2.2.1 Request

The XML encoding of the GetTile operation request shall be a GetTile element conforming to the following XML schema:

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
      xmlns:wmts=”http://www.opengis.net/wmts/1.0”
      xmlns:ows=”http://www.opengis.net/ows/1.1”
      targetNamespace=”http://www.opengis.net/wmts/1.0”
      elementFormDefault=”qualified”
      attributeFormDefault=”unqualified”>
   <element name="GetTile">
      <complexType>
         <complexContent>
            <extension base="ows:RequestBaseType">
               <sequence>
                  <element name="Layer" type="string"/>
                  <element name="Style" type="string" minOccurs="0"/>
                  <element name="Format" type="ows:MimeType"/>
                  <element ref="wmts:Dimension" minOccurs="0"
                        maxOccurs="unbounded"/>
                  <element name="TileMatrixSet" type="string"/>
                  <element name="TileMatrix" type="string"/>
                  <element name="TileRow" type="nonNegativeInteger"/>
                  <element name="TileCol" type="nonNegativeInteger"/>
               </sequence>
               <attribute name="service" type="string" use="required"
                     fixed="WMTS"/>
               <attribute name="version" type="string" use="required"
                     fixed="1.0.0"/>
            </extension>
         </complexContent>
      </complexType>
   </element>

   <element name="Dimension">
      <complexType>
         <simpleContent>
            <extension base="string">
               <attribute name="name" type="string" use="required"/>
            </extension>
         </simpleContent>
      </complexType>
   </element>
</schema>
```

Listing 4: Schema of XML-encoded GetTile request

An example is presented in Listing 5:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetTile service="WMTS" version="1.0.0"
      xmlns="http://www.opengis.net/wmts/1.0">
   <Layer>coastlines</Layer>
   <Style>blue</Style>
   <Format>image/png</Format>
   <Dimension name="TIME">2007-06</Dimension>
   <TileMatrixSet>coastlinesInCrs84</TileMatrixSet>
   <TileMatrix>5e6</TileMatrix>
   <TileRow>42</TileRow>
   <TileCol>112</TileCol>
</GetTile>
```

Listing 5: Example XML-encoded GetTile request

5.2.2.2 Successful response

The response of a successful XML-encoded GetTile operation request shall be an image with the MIME type specified by the Format parameter of the request.

5.2.2.3 Exception response

The response of an unsuccessful XML-encoded GetTile operation request shall be an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3].

5.2.3 GetFeatureInfo

5.2.3.1 Request

The XML encoding of the GetFeatureInfo operation request shall be a GetFeatureInfo element conforming to the following XML schema:

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
      xmlns:wmts=”http://www.opengis.net/wmts/1.0”
      xmlns:ows=”http://www.opengis.net/ows/1.1”
      targetNamespace=”http://www.opengis.net/wmts/1.0”
      elementFormDefault=”qualified”
      attributeFormDefault=”unqualified”>
   <element name="GetFeatureInfo">
      <complexType>
         <complexContent>
            <extension base="ows:RequestBaseType">
               <sequence>
                  <!-- the corresponding GetTile request -->
                  <element ref="wmts:GetTile"/>
                  <element name="J" type="nonNegativeInteger"/>
                  <element name="I" type="nonNegativeInteger"/>
                  <element name="InfoFormat" type="ows:MimeType"/>
               </sequence>
               <attribute name="service" type="string" use="required"
                     fixed="WMTS"/>
               <attribute name="version" type="string" use="required"
                     fixed="1.0.0"/>
            </extension>
         </complexContent>
      </complexType>
   </element>
</schema>
```

Listing 6: Schema of XML-encoded GetFeatureInfo request

An example is presented in Listing 7:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetFeatureInfo service="WMTS" version="1.0.0"
     xmlns="http://www.opengis.net/wmts/1.0">
   <GetTile service="WMTS" version="1.0.0">
      <Layer>coastlines</Layer>
      <Style>blue</Style>
      <Format>image/png</Format>
      <Dimension name="TIME">2007-06</Dimension>
      <TileMatrixSet>coastlinesInCrs84</TileMatrixSet>
      <TileMatrix>5e6</TileMatrix>
      <TileRow>42</TileRow>
      <TileCol>112</TileCol>
   </GetTile>
   <J>23</J>
   <I>35</I>
   <InfoFormat>text/html</InfoFormat>
</GetFeatureInfo>
```

Listing 7: Example XML-encoded GetFeatureInfo request

5.2.3.2 Successful response

The response of a successful XML-encoded GetFeatureInfo operation request shall be a document with the MIME type specified by the InfoFormat parameter of the request.

5.2.3.3 Exception response

The response of an unsuccessful XML-encoded GetFeatureInfo operation request shall be an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3].

6 WMTS SOAP interface

6.1 Declaration of support

A WMTS server that supports SOAP encodings may declare such support by means of the OperationsMetadata section of its Capabilities document. As per Clause 8.3.2 of [OGC 05-009r3], this declaration is accomplished by specifying “SOAP” as an allowed value of the "PostEncoding" constraint within the DCP/HTTP/Post element of each operation that has support for the SOAP encoding. An example is presented in Listing 8:

```
<Operation name="GetTile">
   <DCP>
      <HTTP>
         <Post xlink:href="wmtsBaseUrl">
            <Constraint name="PostEncoding">
               <AllowedValues>
                  …
                  <Value>SOAP</Value>
                  …
               </AllowedValues>
            </Constraint>
         </Post>
      </HTTP>
   </DCP>
</Operation>
```

Listing 8: Example declaration of support for SOAP interface

If the SOAP encoding of an operation has a different base URL (a.k.a. "connect point") from the other encoding(s) of the operation, then the Post element may be repeated with the appropriate constraint(s) declared for each one.

Support for SOAP encodings is optional in WMTS.

6.2 The SOAP envelope

In the SOAP interface, the XML element that contains the request data or response data of an operation shall be placed in the body of a SOAP envelope as specified in SOAP Version 1.2. An example of a SOAP 1.2 envelope with an empty body is presented in Listing 9:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
   </soap:Header>
   <soap:Body>
   </soap:Body>
</soap:Envelope>
```

Listing 9: Example empty SOAP 1.2 envelope

In general (but not always), the soap:Body element contains the XML encodings of the WMTS operation request data or response data as specified in Clause 5. The optional soap:Header element is reserved for optional elements that are beyond the scope of the WMTS specification, such as identity tokens or licensing information.

As recommended by [OGC 08-009r1], the document/literal style shall be used for the SOAP bindings of the operation requests.

6.3 Exception reports

As described in [OGC 06-094r1], if a WMTS server detects an error while processing an operation request encoded in a SOAP envelope, the server shall generate a SOAP 1.2 response message where the content of the Body element is a Fault element containing an ExceptionReport element (as defined in Clause 8.5 of [OGC 06-121r3]). This shall be done using the following XML fragment:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <soap:Fault>
         <soap:Code>
            <soap:Value>soap:Server</soap:Value>
         </soap:Code>
         <soap:Reason>
            <soap:Text>A server exception was encountered.</soap:Text>
         </soap:Reason>
         <soap:Detail>
            <ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1">
               …
            </ows:ExceptionReport>
         </soap:Detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Listing 10: XML fragment of SOAP-wrapped exception report

The Value element (child of the Code element) shall have the value "soap:Server" (where “soap” stands for an arbitrary namespace prefix referring to the SOAP 1.2 namespace “http://www.w3.org/2003/05/soap-envelope”) indicating that this is a server exception. The Text element (child of the Reason element) shall have the value "A Server exception was encountered.". This fixed string is used since the details of the exception shall be specified in the Detail element using an ows:ExceptionReport element.

NOTE: The majority of this clause was copied verbatim from [OGC 06-094r1].

6.4 Operation encodings

6.4.1 GetCapabilities

6.4.1.1 Request

The SOAP encoding of the GetCapabilities operation request shall be a SOAP envelope (see Clause 6.2) whose body contains a GetCapabilities element conforming to the schema defined in Clause 5.2.1.1. An example is presented in Listing 11:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <GetCapabilities service="WMTS"
            xmlns="http://www.opengis.net/ows/1.1">
         <AcceptVersions>
            <Version>1.0.0</Version>
         </AcceptVersions>
         <AcceptFormats>
            <OutputFormat>text/xml</OutputFormat>
         </AcceptFormats>
      </GetCapabilities>
   </soap:Body>
</soap:Envelope>
```

Listing 11: Example SOAP-encoded GetCapabilities request

6.4.1.2 Successful response

The response of a successful SOAP-encoded GetCapabilities operation request shall be a SOAP envelope (see Clause 6.2) whose body contains a Capabilities element as defined in Clause 7.2.3 of [OGC 07-057r6]. An example is presented in Listing 12:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <Capabilities version="1.0.0"
            xmlns="http://www.opengis.net/wmts/1.0.0"
            xmlns:ows="http://www.opengis.net/ows/1.1"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:gml="http://www.opengis.net/gml">
         …
      </Capabilities>
   </soap:Body>
</soap:Envelope>
```

Listing 12: Example SOAP-encoded GetCapabilities response

6.4.1.3 Exception response

The response of an unsuccessful SOAP-encoded GetCapabilities operation request shall be a SOAP exception report document as defined in Clause 6.3.

6.4.2 GetTile

6.4.2.1 Request

The SOAP encoding of the GetTile operation request shall be a SOAP envelope (see Clause 6.2) whose body contains a GetTile element conforming to the schema defined in Clause 5.2.1.1. An example is presented in Listing 13:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <GetTile service="WMTS" version="1.0.0"
            xmlns="http://www.opengis.net/wmts/1.0.0">
         <Layer>coastlines</Layer>
         <Style>blue</Style>
         <Format>image/png</Format>
         <Dimension name="TIME">2007-06</Dimension>
         <TileMatrixSet>coastlinesInCrs84</TileMatrixSet>
         <TileMatrix>5e6</TileMatrix>
         <TileRow>42</TileRow>
         <TileCol>112</TileCol>
      </GetTile>
   </soap:Body>
</soap:Envelope>
```

Listing 13: Example SOAP-encoded GetTile request

6.4.2.2 Successful response

The response of a successful SOAP-encoded GetTile operation request shall be a SOAP envelope (see Clause 6.2) whose body contains an encoding of an image. There are three cases:

1. If the image format specified by the Format element of the request is a well-formed XML element (such as an SVG fragment), then the body of the SOAP envelope shall contain an image in that format.

2. Otherwise, if the format specified by the Format element of the request is a non-XML text document, then the body of the SOAP envelope shall contain a TextPayload element conforming to the schema specified in Listing 14. Within the TextPayload element, the Format element shall contain the MIME type that was specified in the Format element of the request, and the TextContent element shall contain an image in that MIME type. The text needs to be protected appropriately by wrapping it in a CDATA block or by making judicious use of XML character entities (e.g., &lt; and &gt;).

3. Otherwise, the body of the SOAP envelope shall contain a BinaryPayload element conforming to the schema specified in Listing 15. Within the BinaryPayload element, the Format element shall contain the MIME type that was specified in the Format element of the request, and the BinaryContent

element shall contain an image in that MIME type, further encoded in base64. An example is presented in Listing 16.

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
      xmlns:ows=”http://www.opengis.net/ows/1.1”
      targetNamespace=”http://www.opengis.net/wmts/1.0”
      elementFormDefault=”qualified”
      attributeFormDefault=”unqualified”>
   <element name="TextPayload">
      <complexType>
         <sequence>
            <element name="Format" type="ows:MimeType"/>
            <element name="TextContent" type="string"/>
         </sequence>
      </complexType>
   </element>
</schema>
```

Listing 14: Schema of TextPayload element

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
      xmlns:ows=”http://www.opengis.net/ows/1.1”
      targetNamespace=”http://www.opengis.net/wmts/1.0”
      elementFormDefault=”qualified”
      attributeFormDefault=”unqualified”>
   <element name="BinaryPayload">
      <complexType>
         <sequence>
            <element name="Format" type="ows:MimeType"/>
            <element name="BinaryContent" type="base64Binary"/>
         </sequence>
      </complexType>
   </element>
</schema>
```

Listing 15: Schema of BinaryPayload element

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <wmts:BinaryPayload xmlns:wmts=”http://www.opengis.net/wmts/1.0”>
         <wmts:Format>image/png</wmts:Format>
         <wmts:BinaryContent> <!-- base64-encoded -->
            R0lGODdh4AEOAfYAAOGW/9aM9MuC6L943LRu
            0KlkxJ1auJJQrIdGoXs8lXAyiWUofVkecU4U
            …
            Ah0ianvIh+7Fb38oehcBI4NIiPdXhECyf4zY
            iGNnOq7FcfZTiUJ1hfyjVCW3bJ3IiYEAADs=
         </wmts:BinaryContent>
      </wmts:BinaryPayload>
   </soap:Body>
</soap:Envelope>
```

Listing 16: Example SOAP-encoded GetTile response with binary content

(The base64Binary type and associated base64-encoding rules are defined in the *XML Schema Part 2* W3C recommendation.)

It has been decided that no specific transfer-encoding optimization mechanisms (such as MTOM/XOP or Fast Infoset) should be dictated at this time. However, use of gzip transfer encoding is encouraged.

6.4.2.3 ExceptionResponse

The response of an unsuccessful SOAP-encoded GetTile operation request shall be a SOAP exception report document as defined in Clause 6.3.

6.4.3 GetFeatureInfo

6.4.3.1 Request

The SOAP encoding of the GetFeatureInfo operation request shall be a SOAP envelope (see Clause 6.2) whose body contains a GetFeatureInfo element conforming to the schema defined in Clause 5.2.3.1. An example is presented in Listing 17:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <GetFeatureInfo service="WMTS" version="1.0.0"
            xmlns="http://www.opengis.net/wmts/1.0.0">
         <GetTile service="WMTS" version="1.0.0">
            <Layer>coastlines</Layer>
            <Style>blue</Style>
            <Format>image/png</Format>
            <Dimension name="TIME">2007-06</Dimension>
            <TileMatrixSet>coastlinesInCrs84</TileMatrixSet>
            <TileMatrix>5e6</TileMatrix>
            <TileRow>42</TileRow>
            <TileCol>112</TileCol>
         </GetTile>
         <J>23</J>
         <I>35</I>
         <InfoFormat>text/html</InfoFormat>
      </GetFeatureInfo>
   </soap:Body>
</soap:Envelope>
```

Listing 17: Example SOAP-encoded GetFeatureInfo request

6.4.3.2 Successful response

The response of a successful SOAP-encoded GetFeatureInfo operation request shall be a SOAP envelope (see Clause 6.2) whose body contains an encoding of a feature description. There are three cases:

1. If the format specified by the InfoFormat element of the request is a well-formed XML element, (such as a GML or XHTML fragment), then the body of the SOAP envelope shall contain a feature description in that format.

2. Otherwise, if the format specified by the InfoFormat element of the request is a non-XML text document (such as text/plain or text/html), then the body of the SOAP envelope shall contain a TextPayload element conforming to the schema specified in Listing 14. Within the TextPayload element, the Format element shall contain the MIME type that was specified in the InfoFormat element of the request, and the TextContent element shall contain a feature description in that MIME type. The text needs to be protected appropriately by wrapping it in a CDATA block or by making judicious use of XML character entities (e.g., &lt; and &gt;). An example is presented in Listing 18.

3. Otherwise, the body of the SOAP envelope shall contain a BinaryPayload element conforming to the schema specified in Listing 15. Within the BinaryPayload element, the Format element shall contain the MIME type that was specified in the InfoFormat element of the request, and the BinaryContent element shall contain a feature description in that MIME type, further encoded in base64.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <wmts:TextPayload xmlns:wmts="http://www.opengis.net/wmts/1.0">
         <wmts:Format>text/html</wmts:Format>
         <wmts:TextContent>
            <![CDATA[
               <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
               <HTML>
                  …
               </HTML>
            ]]>
         </wmts:TextContent>
      <wmts:TextPayload>
   </soap:Body>
</soap:Envelope>
```

Listing 18: Example SOAP-encoded GetFeatureInfo response with HTML content

6.4.3.3 Exception response

The response of an unsuccessful SOAP-encoded GetFeatureInfo operation request shall be a SOAP exception report document as defined in Clause 6.3.

6.5 WSDL document

A WMTS server may publish one or more WSDL 1.1 documents describing its SOAP interface. How these documents are published is up to the discretion of the server, but it is recommended by [OGC 08-009r1] that they be published through CSW registries. This approach follows the publish-find-bind pattern and this allows humans and services to discover the WSDL document in a standardized manner.

In a future version of the WMTS specification, one that is based on OWS Common 1.2 rather than OWS Common 1.1, a <WSDL> element which specifies a reference to the service's WSDL document will be added to the Capabilities document. This will make it possible for a client of a SOAP-enabled WMTS server to discover the server's WSDL document via the GetCapabilities operation. (The reverse approach is already possible; given a WMTS server's WSDL document, a client can discover how to make the appropriate GetCapabilities request in order to retrieve the server's Capabilities document.)

7 WMTS REST interface

7.1 Overview

The WMTS REST interface is a well-defined set of canonical URIs that can be used to access the resources served by a WMTS server. The three types of resources served by a WMTS server are the

Capabilities document, the tiles and (optionally) one or more feature-info documents describing every pixel of every tile. These resource types correspond to the three traditional WMTS operations GetCapabilities, GetTile and GetFeatureInfo. Tiles and feature descriptions are algorithmic resources, in that they are specified as being generated by an algorithm with certain input parameters. The representations of all WMTS resources are not time-dependent and are guaranteed to never change over a long period of time, and therefore can be safely cached by the server, by the client, or by any intermediary.

Unlike the other interfaces, in which a request can take an arbitrary number of forms, the URIs defined by the REST interface are rigid in that (with the possible exception of server aliases) there is a one-to-one mapping between a WMTS URI and the resource it references. This is achieved by specifying URIs with a path-based internal structure, where the path elements are the parameters of an algorithm that generates the resource referenced by that URI. That algorithm, in turn, corresponds to a traditional WMTS operation (GetTile and GetFeatureInfo). Query parameters (as in traditional KVP) are not used.

The above implies that, at a higher level of abstraction, there is currently a one-to-one mapping between a WMTS REST resource and the result of invoking a traditional WMTS operation with certain input data. Note, however, that this one-to-one mapping between the REST interface and the procedural interfaces of WMTS is accidental and derives from our having taken a conservative approach to specifying the set of resource types for WMTS. It is quite possible that future versions of WMTS will specify a larger set of resource types in order to achieve better conformance with the principles of REST and to maximize the benefits (e.g., to support navigation and search).

One benefit of this overall approach is that it allows for simple and efficient server-side and/or client-side caching of tiles. Another benefit is that it allows a WMTS server to be implemented as a simple file system populated with pre-generated tiles in the appropriate directory structure.

Future versions of this specification, with guidance from future versions of the OWS Common standard and/or best-practices papers, may further formalize the RESTful nature of this interface.

If a WMTS server receives a GET request whose URI is not recognized, it shall return an HTTP response message with an error code of “404 Not Found” whose body, where possible, contains an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3].

7.2 Declaration of support

A WMTS server shall declare support for the REST interface by means of the OperationsMetadata section of its Capabilities document. Following the guidance of Clause 8.3.2 of [OGC 05-009r3], this declaration is accomplished by specifying “RESTful” as an allowed value of the "GetEncoding" constraint within the DCP/HTTP/Get element of each traditional WMTS operation that corresponds to the retrieval of a resource (capabilities, tile, or feature description) in the WMTS REST interface. An example is presented in Listing 19:

```
<Operation name="GetTile">
   <DCP>
      <HTTP>
         <Get xlink:href="wmtsBaseUrl">
            <Constraint name="GetEncoding">
               <AllowedValues>
                  …
                  <Value>RESTful</Value>
                  …
               </AllowedValues>
            </Constraint>
         </Get>
      </HTTP>
   </DCP>
</Operation>
```

Listing 19: Example declaration of support for REST interface

If the URI of a resource in the REST interface has a different base URL (a.k.a. "connect point") from the other encoding(s) of the corresponding operation, then the Get element shall be repeated with the appropriate constraint(s) declared for each one.

Support for the REST interface is optional.

7.3 Resource types and canonical URIs

7.3.1 Capabilities resource

7.3.1.1 Canonical URI scheme

A capabilities resource shall have a URI of the following form:

{wmtsBaseUrl}/*{wmtsVersion}*/WMTSCapabilities*{formatExtension}*

where *{wmtsBaseUrl}* is the base URL of the GetCapabilities operation with the “GetEncoding” constraint value of “RESTful” (as advertised in the Capabilities document), *{wmtsVersion}* is the specification version and *{formatExtension}* is the filename extension that identifies the desired representation of the capabilities resource. A *{formatExtension}* of “.xml” indicates the standard Capabilities document as defined in Clause 7.2.3 of [OGC 07-057r6]. No other representation formats are currently defined by the WMTS standard, but vendor-specific formats may exist.

The client-server version negotiation mechanism is as follows. The client requests the Capabilities document with the most-preferred WMTS version. If an HTTP 404 error is returned, the client can try again specifying the next-preferred WMTS version, and ultimately give up if it is determined that no Capabilities document exists for any of the client-supported WMTS versions.

NOTE: This version-negotiation mechanism differs from that of the other GetCapabilities operation request encodings, which include the complete list of client-acceptable versions as part of the URL or content body of the request. This approach is used here in order to align with REST principles and practices.

An example URL for a capabilities resource is presented in Listing 20:

```
http://www.maps.cat/tiledWorld/1.0.0/WMTSCapabilities.xml
```

Listing 20: Example capabilities resource URI

7.3.1.2 Successful response

The response of a successful capabilities operation recource retrieval request (GET) shall be an HTTP response message carrying a Capabilities document of the specified WMTS version and representation format, where a specified representation format of ".xml" indicates the standard Capabilities document as defined in Clause 7.2.3 of [OGC 07-057r6].

7.3.1.3 Exception response

The response of an unsuccessful capabilities resource retrieval request (GET) shall be an HTTP response message with the appropriate error code whose body, where possible, contains an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3]. Determination of what error conditions map to which HTTP error codes is up to the discretion of the server.

7.3.2 Tile resource

7.3.2.1 Canonical URI scheme

A tile resource shall have a URI of the following form:

{wmtsBaseUrl}/*{layer}*/*{style}*/*{dimension1}*/.../*{dimensionN}*/*{TileMatrixSet}*/*{TileMatrix}*/*{TileRow}*/*{TileCol}{formatExtension}*

where {wmtsBaseUrl} is the base URL of the GetTile operation with the "GetEncoding" constraint value of "RESTful" (as advertised in the Capabilities document), *{formatExtension}* is the format extension that identifies the representation of the tile image (as advertised in the Capabilities document), and the other path elements correspond to the GetTile operation request parameters of the same name in Table 17 of [OGC 07-057r6]. The service, version and request parameters are not explicitly encoded in the URI. The service and version parameters are assumed to be implied as necessary by *{baseUrl}*, and the request parameter is not relevant in the REST interface (it is essentially implied by the structure of the URI).

An example URI for a tile resource is presented in Listing 21 (where "http://www.maps.cat/tiledWorld/1.0.0" is what was advertised by the Capabilities document as the "RESTful" base URL of the GetTile operation):

```
http://www.maps.cat/tiledWorld/1.0.0/coastlines/blue/2007-06/
     coastlinesInCrs84/5e6/42/112.png
```

Listing 21: Example tile resource URI

7.3.2.2 Successful response

The response of a successful tile resource retrieval request (GET) shall be an HTTP response message carrying image with the MIME type implied by the *{formatExtension}* in the URI of the resource.

7.3.2.3 Exception response

The response of an unsuccessful tile resource retrieval request (GET) shall be an HTTP response message with the appropriate error code whose body, where possible, contains an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3]. Determination of what error conditions map to which HTTP error codes is up to the discretion of the server. If the error condition is due to a malformed request or a request with illegal path values, the HTTP error code returned should be "404 Not Found".

7.3.3 Feature description resource

7.3.3.1 Canonical URI scheme

A feature description resource shall have a URI of the following form:

{wmtsBaseUrl}/*{layer}*/*{style}*/*{dimension1}*/.../*{dimensionN}*/*{TileMatrixSet}*/*{TileMatrix}*/*{TileRow}*/*{TileCol}*/*{J}*/*{I}{infoFormatExtension}*

where {wmtsBaseUrl} is the base URL of the GetTile operation with the "GetEncoding" constraint value of "RESTful" (as advertised in the Capabilities document), *{infoFormatExtension}* is the format extension that identifies the representation of the feature description (as advertised in the Capabilities document), and the other path elements correspond to the GetFeatureInfo operation request parameters of the same name in Table 19 of [OGC 07-057r6]. The service, version and request parameters are not explicitly encoded in the URI. The service and version parameters are assumed to be implied as necessary by *{baseUrl}*, and the request parameter is not relevant in the REST interface (it is essentially implied by the structure of the URI).

An example URI for a feature description resource is presented in Listing 22 (where "http://www.maps.cat/tiledWorld/1.0.0" is what was advertised by the Capabilities document as the "RESTful" base URL of the GetFeatureInfo operation):

```
http://www.maps.cat/tiledWorld/1.0.0/coastlines/blue/2007-06/
     coastlinesInCrs84/5e6/42/112/23/35.html
```

Listing 22: Example feature description resource URI

7.3.3.2 Successful response

The response of a successful feature description resource retrieval request (GET) shall be an HTTP response message carrying a document with the MIME type implied by the *{infoFormatExtension}* in the URI of the resource.

7.3.3.3 Exception response

The response of an unsuccessful feature description resource retrieval request (GET) shall be an HTTP response message with the appropriate error code whose body, where possible, contains an XML ExceptionReport document as defined in Clause 8.5 of [OGC 06-121r3]. Determination of what error conditions map to which HTTP error codes is up to the discretion of the server. If the error condition is due to a malformed request or a request with illegal path values, the HTTP error code returned should be “404 Not Found”.

8 Future work and miscellaneous considerations

8.1 Optimization of SOAP-wrapped GetTile response

It may be desirable in the future to provide guidance on, or even to specify explicitly, what optimization mechanism(s), if any, shall be utilized during the encoding and/or transmission of a SOAP-wrapped GetTile response.

8.2 JSON encoding of Capabilities document

During the course of OWS-6 DSS, a suggestion was made (by Wittaker Mathot of ESRI) that a JSON encoding of the WMTS Capabilities document should also be specified. The justification for this suggestion is twofold. Firstly, by communicating in native JavaScript syntax, it would make it easier for JavaScript clients to work with WMTS services. Secondly, it would help solve the common issue of cross-domain JavaScript communication (the typical solution of which is to introduce a proxy service).

It was agreed that should such an encoding be specified for the Capabilities document, it would be specified as a simple set of rules defining a transformation from the XML schemas, or even from the UML diagrams themselves, rather than introducing an alternate structure. Furthermore, the intent would not be for the JSON encoding to replace the XML encoding, but merely to exist as a companion encoding in order to provide JavaScript clients with a more JavaScript-friendly interface.

The specification of such an encoding is beyond the scope of OWS-6. Furthermore, it is unlikely to be addressed in the first version of the WMTS specification (beyond allowing the ability for alternate encodings of the Capabilities document to exist). There is some concern that introducing another encoding may increase server implementation effort and decrease interoperability, and this needs to be considered. Finally, if a JSON encoding of the Capabilities document is deemed to be worthy of specification, its utility likely extends beyond the realm of WMTS, so it should therefore perhaps be ultimately specified by OWS Common so that all OGC services can potentially make use of it.

8.3 Extending the REST interface

The WMTS REST interface defined by this engineering report is merely the starting point for a potentially rich set of fully-transactional operations. Several suggestions have been made during the course of OWS-6 DSS as to how we should consider evolving the WMTS REST interface beyond what has been defined by this engineering report. For example, Peter Rushforth of GeoConnections has suggested the addition of a PUT operation for tile resources, allowing (appropriately authorized) clients to create tiles. Conceivably this concept could be extended even further by introducing the ability for WMTS clients to PUT, POST or DELETE entire tile matrix sets. However, the details of such operations have not yet been explored.

Pat Cappelaere of Vightel Corporation has suggested the following steps for moving forward:

- First step is to define the data model. I would love to see that clearly articulated in the document. I do suspect however that it is there but it did not jump at me.

- The next thing to think about (at the system level for perfect harmony) is transaction. How would you add a resource, edit a resource and delete a resource? Are we going to create (again) our own protocol or use an existing one? This certainly has to apply to a WMTS.

- Step 3 is to think about the discovery of metadata. Are we going to create (again) our own protocol or use an existing one?

- Step 4 is to think about search. Are we going to create (again) our own protocol or use an existing one?

- Step 5 is data publishing. Are we going to create (again) our own protocol or use an existing one?

- Step 6 is security. Are we going to create (again) our own protocol or use an existing one?

One thing is certain. The current structure of the WMTS Capabilities document (bound by the rules of [OGC 06-121r3]) is not sufficiently able to represent the operations of a fully-transactional REST interface. The mechanism defined by this engineering report treats each REST operation as simply another encoding of an operation in an existing procedure-oriented service definition. This is sufficient for defining a read-only REST interface where the only resources defined are those returned by the existing procedure-oriented service operations. (And thus it's a reasonable approach for the initial introduction of a REST interface on top of a traditionally service-oriented software system.) However, it won't suffice for a richer set of REST operations. It may be necessary to define an entirely different form of “capabilities” mechanism specifically for communication of the resource-oriented capabilities of the service, and leave the traditional Capabilities document alone to define the procedure-oriented service capabilities. This problem, of course, is not limited to WMTS; it's a general problem that will need to be resolved for any OGC service requiring the addition of a REST interface. Perhaps this is a topic for OWS-7 to explore.

Annex A – WMTS Implementation Performance Considerations

A.1 Introduction

This annex is an overview of the performance considerations when implementing WMTS. Much of the content is relevant to web map services in general.

This annex was submitted by Jim Groffen of LISAsoft.

A.2 HTTP Cache-Control Header

The ability to cache responses to HTTP requests is an intrinsic feature of HTTP. The two main reasons for caches are:

- Reduce response times by caching a possibly expensive generation of the response by the origin server.

- Reduce network traffic by eliminating the need for a request / response to traverse to the origin server.

Caching requires context information for the cached content provided by the origin server. This is provided by the HTTP header "Cache-Control". Without this header the downstream cache points are not provided with context of the content and cannot make an informed decision on caching for that content.

In the following example of a Cache-Control header, we specify that the cached content is public (not intended for a single user), is considered valid for five days (max-age is in seconds) and must strictly follow the caching rules set by the origin server.

```
Cache-Control: public, max-age=432000, must-revalidate
```

For volatile information it is recommended that the "no-cache" setting is specified. The no-cache directive tells all downstream caches to cache the content but never release the content without first validating it against the origin server. This check usually entails checking the Last-Modified header specified by the origin server against the cached copy.

A.2.1 Etag Header – Entity Tags

There is a new mechanism for validation of a resource as of HTTP 1.1 called ETag. Instead of relying on a date which relies on the synchronization of server clocks, an ETag header can be specified. This header is a unique identifier that is changed by the content server each time the content changes.

A.2.2 Other Relevant Headers

- Age: The length of time in seconds since the response was generated at the origin server. According to the HTTP 1.1 specification if the responding service has a cache then it must include an age header.

- Expires: As an alternative to max-age you can explicitly set a date for the expiry of the content, e.g. Fri, 13 Mar 2009 11:55:45 GMT. Both Expires and max-age options should not be specified but if they are, max-age should override Expires even if expires is more restrictive.

- Last-Modified: Set when the content last changed. In combination with no-cache this can be used to verify that the content has been altered since it was cached.

- Content-Length: This header is also sometimes used with no-cache to detect changed content, but is less reliable and only recommended when attempting to make cache decisions in the absence of cache control information. It is still highly recommended that this header is set by your service though, as it is used in HTTP connection persistence support.

As a warning: the Pragma "no-cache" HTTP header is still sometimes used but is considered deprecated. Do not use this header for cache control.

A.2.3 Cache Control Server Side Tools

Most web servers provide server level control of caching headers. In apache there is mod_expires and mod_headers. IIS provides header management capabilities and there are packages available that extend this capability. This allows the implementer to separate configuration of Cache-Control, Expires and other headers from the service.

A.3 Connection Persistence

HTTP 1.1 allows for a single TCP connection to service multiple requests. Connection persistence is supported in most browsers and enabled on web services by default. Most implementations of connection persistence are dependent on knowing the length of a response before response generation and is less useful for dynamically generated content, but relevant to a WMTS service.

To cater for connection persistence, ensure Content-Length headers are set in server responses. Most web servers allow for configuration of the connection persistence support. For example Apache Keep-Alive Support honors maximum number of requests and connection timeout configuration.

A.3.1 Maximum HTTP Connections Limitation

HTTP 1.1 stipulates that no more than two connections should be kept alive to the same server from the same client simultaneously. Most browsers honor this, which limits the number of simultaneous requests for tiles that can occur between client and server.

A common workaround for this in AJAX based clients is to configure multiple host names to the one service. OpenLayers supports this workaround by providing multiple service names for the one service. Keep in mind that most browsers will by default limit the number of active HTTP connections to 4 or 6, though this is (not easily) configurable.

While the above is an example of tricking a browser into maintaining more than two connections to a single server, as WMTS is implemented to support disk-only caching distribution of the cache across multiple services is feasible.

A.4 Tiled Service Optimizations

This section details optimizations that are specific to tiled services or a particular implementation such as:

- GeoWebCache: developed in Java, incorporated into GeoServer.
- TileCache: developed in Python.

A.4.1 GeoWebCache Server Configuration

GeoWebCache runs in a Java Web Service container such as, JBOSS, Jetty and Tomcat. The JDK, JRE, Java container and service configuration all affect GeoWebCache performance. The GeoServer user manual contains a section that covers installation of GeoServer in a production environment [6], which addresses the above.

There are also various articles [3] [4] [12] that discuss optimization of GeoServer which are relevant to GeoWebCache.

A.4.2 TileCache Server Configuration

The Mechanisms for exposing Python code as a web service are varied. TileCache supports the following:

- CGI: Most web services support Python as a CGI scripting language. TileCache can be used this way but it is not recommended as CGI scripting starts a new process for each request so TileCache configuration and so on is read every time a request is received. While FastCGI overcomes many of the problems of using CGI, performance tests [9] [11] have shown that this is still slower than the following alternatives.

- mod_python: This Apache module integrates Python into the Apache server.
- modwsgi: Web Server Gateway Interface defines an interface between web servers and web applications for Python. This standard allows Web Servers to support a single method for Python integration. Python modules need only support WSGI instead of a plethora of web service integration methods and all WSGI supporting web services are supported by this application.

For Apache, Performance of mod_python and modwsgi is comparable when used with TileCache. The decision is reduced to: modwsgi is more performant but mod_python provides a richer integration with Apache. The opinion of the author is that WSGI will overtake mod_python as a preferred solution for Python in general.

An MIT Licensed ISAPI WSGI filter is available on Google Code. Though untested, this would be the preferred solution for using TileCache with IIS. CGI has been used to host TileCache on IIS with the usual performance drawbacks.

A.4.3 Metatiling

Metatiling is an important capability for tiled map services. Instead of generating each tile individually a "metatile" is generated. This is a single large map image that is divisible into a number of tiles. Both TileCache and GeoWebCache [6] support metatiling.

Metatiling avoids duplicate labeling. As tiles are generally smaller than the map pane of a client consuming the tiles the labels could be duplicated once per tile. Metatiling overcomes this problem.

Generating one 512 x 512 image is almost always going to be faster than generating four 256 x 256 images. Generation of tiles from the originating WMS service will involve accessing source data (database, shape files, image files) once instead of four times, in most cases involving a similar set of data.

A limiting factor when deciding on a metatile size is the amount of memory a large metatile takes for a service to build.

A.4.4 Web Map Service Optimization / Replication

All image caches source their imagery at some point by requesting the image from the originating service. In this case the originating service will usually be a WMS service. Generation of tiles is heavily dependent on the performance of the source WMS service and optimization of the originating service is recommended [3] [4] [12]. No matter how much a map server is optimized it will not be able to generate tiles as quickly as a Tile Server can consume them.

A sure way to increase tile throughput is to replicate the WMS service to multiple machines. While some tiling services allow for a list of WMS host names to be specified to support this, it can instead be achieved without modification to the tile service configuration with load balancers.

A.5 Tailoring Tiles for the Intended Audience

The benefit of a tiled map service is dependent on ensuring the tiles generated by the service are appropriate for the clients that use them. The following is a list of possible situations and the types of tiles that best suit that situation. These suggestions are based on real-world situations and feedback from consumers of tiles encountered by LISAsoft.

Situation	**Recommendation**
Map coverage required by clients is small to moderately sized regions of complex, slow to generate tiles.	Use a pre-seeding tool to proactively seed the cache with generated tiled instead of as the user requests them.
Some of the data to be tiled is volatile, changing frequently.	Isolate the volatile data into a separate layer of tiles with transparency. This allows you to control the caching rules of the two layers independently.
Tiles are often consumed by mobile phones / devices.	Use PNG format for tiles. PNG support is more comprehensive than other image formats on mobile phones.
Tiles are to be accessed from a portable device such as a mobile or PDA.	Most portable devices have a smaller display than a standard computer so a natural conclusion would be to build smaller tiles. In practice, depending on the device, file systems on portable devices are slower (longer seek times) and do not handle complex folder structures or many files well. The consuming application may also be optimized to access one large spatially referenced image as opposed to many small images. The recommendation is trial the tiles on the target platform, catering to that particular platform.

If there are multiple combinations of users and disk space is not an issue, a viable strategy is to configure one or more layers of the same data to accommodate the varying types of users.

A.6 References

[1] Ajax Performance: Circumventing Browser Connection Limits for Fun and Profit:
http://www.ajaxperformance.com/2006/12/18/circumventing-browser-connection-limits-for-fun-and-profit/

[2] Apache Keep-Alive Support:
http://httpd.apache.org/docs/1.3/keepalive.html

[3] GeoServer: Optimizing WMS Output:
http://geoserver.org/display/GEOSDOC/5+Optimizing+WMS+output

[4] GeoServer: Performance benchmarking with optimization recommendations:
http://presentations.opengeo.org/2008_FOSS4G/WebMapServerPerformance-FOSS4G2008.pdf

[5] GeoServer: TileCache Tutorial:
http://geoserver.org/display/GEOSDOC/TileCache+Tutorial

[6] GeoServer Users Guide 2.6: GeoServer in Production Environment
http://geoserver.org/display/GEOSDOC/2.6+GeoServer+in+Production+Environment

[7] GeoWebCache: Metatiles:
http://geowebcache.org/trac/wiki/metatiles

[8] Google Code isapi-wsgi Project:
http://code.google.com/p/isapi-wsgi/

[9] Google Code modwsgi Project – Performance estimates:
http://code.google.com/p/modwsgi/wiki/PerformanceEstimates

[10] Mark Nottingham Blog: Caching Tutorial
http://www.mnot.net/cache_docs/

[11] Oegeo Article regarding TileCache performance testing
http://oegeo.wordpress.com/2008/03/17/benchmarking-tilecache-part-1/

[12] Rendering Optimization of GeoTools:
http://geotools.codehaus.org/Rendering+Optimization

[13] W3C HTTP 1.1 Section 8.1: Persistent Connections:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

[14] W3C HTTP 1.1 Section 14: Header Field Definitions:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

[15] Wikipedia HTTP persistent connection:
http://en.wikipedia.org/wiki/HTTP_persistent_connection