The Architecture of qtspecs

The purpose of this document is to provide the members, especially the editors, of the W3C XML Query and XSL Working Groups (collectively “QT”) information needed to comprehend the architecture of the qtspecs CVS tree, the build system, the structure of documents, and the like.

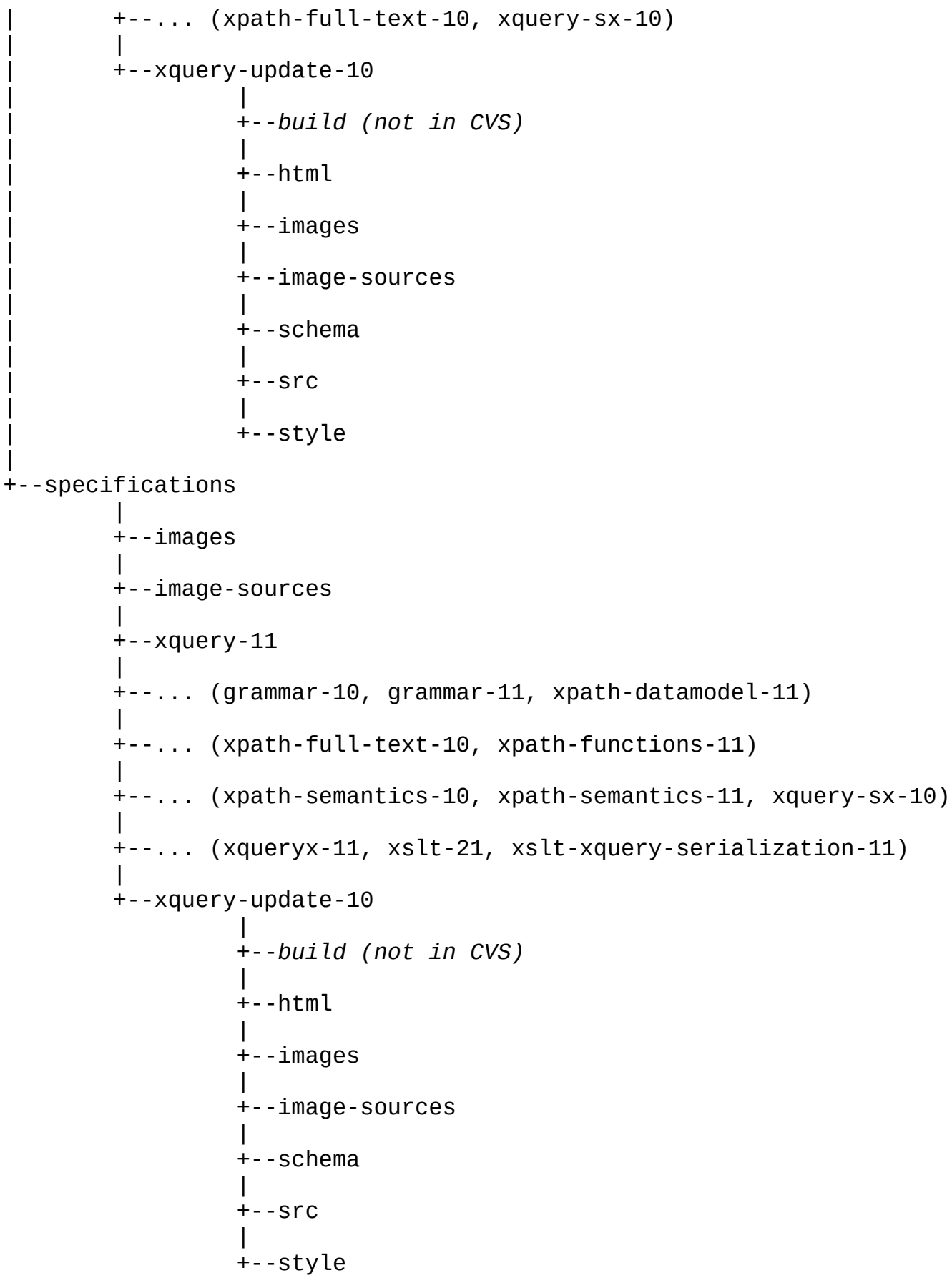
The document has several sections, each attempting to provide and explain the necessary information about some particular aspect of the topics mentioned in the preceding paragraph.

1 Directory Structure of qtspecs

The structure of the directory tree in the CVS tree rooted at cvs.w3.org:/w3ccvs/XML/Group/qtspecs/ has been carefully designed to create an ordered environment that works equally well for all QT specifications. However, the structure can be difficult to grasp without understanding a few facts.

1.1 The directory structure is:

```
qtspecs
  |
  +--etc
  |
  +--lib
  |
  +--schema
  |
  +--style
  |
  +--requirements
  |       |
  |       +--xquery-11
  |       |
  |       +--... (xpath-full-text-10, xquery-sx-10, xpath-21?, xslt-21?)
  |       |
  |       +--xquery-update-10
  |                |
  |                +--build (not in CVS)
  |                |
  |                +--html
  |                |
  |                +--images
  |                |
  |                +--image-sources
  |                |
  |                +--schema
  |                |
  |                +--src
  |                |
  |                +--style
  |
  +--use-cases
  |       |
  |       +--xquery-11
  |       |
```


+--... (xpath-full-text-10, xquery-sx-10)
+--xquery-update-10
+--build (not in CVS)
+--html
+--images
+--image-sources
+--schema
+--src
+--style
+--specifications
+--images
+--image-sources
+--xquery-11
+--... (grammar-10, grammar-11, xpath-datamodel-11)
+--... (xpath-full-text-10, xpath-functions-11)
+--... (xpath-semantics-10, xpath-semantics-11, xquery-sx-10)
+--... (xqueryx-11, xslt-21, xslt-xquery-serialization-11)
+--xquery-update-10
+--build (not in CVS)
+--html
+--images
+--image-sources
+--schema
+--src
+--style

1.2 The intended use of each directory

1.2.1 `qtspecs`

This directory is called the *root directory* for all QTspecs documents.

1.2.2 `qtspecs/etc`

This directory is used to contain a number of files that either do not fit well into any other directory in this tree or that are used for various purposes related to building many documents.

Examples of the former category include the so-called *anchor documents*. There is one anchor document associated with each of the QT specifications (soon to include the requirements and use cases documents, too). The documents provide a cross reference between the id attribute values of each <h1>, <h2>, *etc*. and the number and title of the corresponding headings.

Examples of the latter category include a master ant script that will cause all documents in qtspecs to be built, XML files defining properties used by other ant scripts, a DTD file that defined entities used in constructing documents' status sections, and the like.

1.2.3 `qtspecs/lib`

This directory serves as a code library, containing Java jar files for several XSLT engines, XML parsers, ant, *etc*. It is hoped that this will one day permit editors to build documents without having to install specific versions of XSLT engines.

1.2.4 `qtspecs/schema`

All DTDs that are used by several, or all, of the QT documents are kept in this directory. This is called the *shared schema directory*.

(At the time of writing, a few DTDs used by only a single document are also found in that directory. All such one-document DTDs will be removed from this directory and placed into the …/schema directory associated with its associated document. It is also possible that there are DTDs in the individual documents' schema directories that should be moved into this "shared" style directory.)

1.2.5 `qtspecs/style`

All XSLT stylesheets that are used by several, or all, of the QT documents are kept in this directory. This is called the *shared styles directory*.

(At the time of writing, a number of stylesheets used by only a single document are also found in that directory. All such one-document stylesheets will be removed from this directory and placed into the …/style directory associated with its associated document. It is also possible that there are stylesheets in the individual documents' style directories that should be moved into this "shared" style directory.)

1.2.6 `qtspecs/requirements`

All requirements documents are to be found in subdirectories of this directory. The substructure of this directory is very nearly the same as that of the specifications directory described below.

This directory is called the *requirements root directory*.

1.2.7 `qtspecs/use-cases`

All use cases documents are to be found in subdirectories of this directory. The substructure of this directory is very nearly the same as that of the specifications directory described below.

This directory is called the *use cases root directory*.

1.2.8 qtspecs/specifications

All specification documents, plus grammar files, are located in subdirectories of this directory. Each subdirectory of …/qtspecs/specifications/ is named using the shortname of the document (including the version code of the specific version of the document); for example, the name of the directory for XQuery 1.1 is query-11 and the name of the directory for XQuery Scripting Extensions 1.0 is xquery-sx-10.

This directory is called the *specifications root directory*.

1.2.9 qtspecs/specifications/images

This directory holds the renderable form of all images that are used by multiple specifications. This is called the *shared images directory*.

Note that there is not, at this time, a corresponding subdirectory in the …/requirements/ or …/use-cases/ directories. This is because the Requirements and Use Cases documents do not presently use images.

1.2.10 qtspecs/specifications/image-sources

This directory holds the source (editable) form of all images that are used by multiple specifications. This is called the *shared image sources directory*.

Note that there is not, at this time, a corresponding subdirectory in the …/requirements/ or …/use-cases/ directories. This is because the Requirements and Use Cases documents do not presently use images.

1.2.11 qtspecs/specifications/xquery-11

This directory is used for the XQuery 1.1 specification and its various associated files; all such files, other than the ant script (build.xml) are contained in subdirectories. In this document, it serves as a prototype for the directories used by each document (including the requirements and use-cases documents).

This directory is called the *document root directory*.

1.2.12 qtspecs/specifications/xquery-11/build

This directory is not represented in the CVS system, but is dynamically created in your local system's copy of the qtspecs tree. The ant scripts may, but do not necessarily, use the directory for temporary files that do not need to be checked into CVS but that are useful when building a document.

1.2.13 qtspecs/specifications/xquery-11/html

Most of the results of building a document are placed into this directory. Specifically at least two files are placed here: Overview.html (the document itself in HTML form) and *shortname*.xml.

That latter file deserves some additional explanation, including its name. A given document has a *generic shortname* and a *shortname*. For example, the generic shortname for XQuery 1.1 is "xquery", while the (true) shortname is "xquery-11". Therefore, xquery-11.xml is the filename of the second file stored in this directory.

The content of xquery-11.xml is the "fully assembled" source file for XQuery 1.1. The phrase "fully assembled" means that all of the various source files (*e.g.*, individual XML files for each chapter or appendix) have been assembled into a single unified XML file. (Other assembly may also have been performed, especially inclusion of appropriate versions of grammar files.)

This directory will also contain the result of building any namespace documents required because of namespaces defined by the document itself.

1.2.14 qtspecs/specifications/xquery-11/schema

All DTDs that are used by this specific document are kept in this directory. This is called the *local schema directory*.

1.2.15 qtspecs/specifications/xquery-11/style

All XSLT stylesheets that are used by this specific document are kept in this directory. This is called the *local style directory*.

1.2.16 qtspecs/specifications/xquery-11/images

This directory holds the renderable form of all images that are used by this specific document. This is called the *local images directory*.

1.2.17 qtspecs/specifications/xquery-11/image-sources

This directory holds the source (editable) form of all images that are used by this specific document. This is called the *local image sources directory*.

1.2.18 qtspecs/specifications/xquery-11/src

All XML source files (and, rarely, perhaps other source files) required to build this specific document are kept in this directory. Some documents incorporate "ancillary files" (examples, perhaps) that might be kept in a subdirectory of this directory. That decision is entirely left to the editor(s) of the specific document.

This directory must contain a file whose filename is *genericshortname*.xml, where *genericshortname* is a generic shortname as described in Section 1.2.13. This file is sometimes called the *root file* for the document. For example, the root file for XQuery 1.1 is named xquery.xml. Please note the absence of version information in that filename.

2 The build system

2.1 The basics

The documents maintained in the qtspecs area of CVS are generated using a series of tools collectively known as *the build system*. The most obvious such tool is named "ant". Ant is a tool that is driven by an XML file (called an *ant script*), targets defined in that XML file, and tasks invoked in that XML file. (The default filename of an ant script is "build.xml".) It provides a declarative way of expressing the relationships between various stages of a build that ensures that all changes to files are "noticed" and all tasks required to fulfill targets using those files are invoked to regenerate the output of the targets.

Editors of specific documents are not necessarily expected to create, or to maintain, the ant scripts associated with building their documents. (Indeed, they are discouraged from doing so in general, so that a central point of maintenance has responsibility.)

This document will *not* attempt to document the use of ant or the creation of ant scripts. However, it is worth explaining a handful of important concepts that are crucial in use to build the QT documents.

Some editors will do their work using some variation of the Windows operating system, while others will use some variation of Unix/Linux. To the best of my knowledge, editors working on either system will use some sort of shell (a/k/a "command line") interface that provides a "console".

2.1.1 Targets

The ant scripts used to build QT documents use targets typically named "all", "spec", and several others. (Note: As of this writing, there is great inconsistency between ant scripts for various documents. Those inconsistencies will be resolved as soon as possible.)

The target named "all" does what its name implies: It performs all actions defined in that particular ant script. Analogously, "spec" performs the actions required to build the specific document, but not (necessarily) auxiliary actions.

2.1.2 Tasks

Ant defines a number of built-in tasks (and permits the creation of additional tasks). Some of the tasks used by the QT build system are:

- echo – causes a text message to be displayed on the "console"
- xslt – invokes an XSLT processor to perform a transformation
- xmlvalidate – invokes an XML schema validator
- uptodate – checks whether a target must be built because of changes to files on which it depends

There are many other tasks available in ant, and some of them have many options. Documentation is available for use by anybody interested in pursuing knowledge about ant.

2.1.3 Properties

One of the key characteristics of ant that makes it so beneficial is its use of *properties*. A property in ant is similar in some ways to entities in XML. In particular, a property is given a value and then that property can be used symbolically elsewhere to evoke that value. The QT build.xml ant scripts make heavy use of properties to give symbolic names to directories used in building the QT documents.

For example, the property named "shared.schema.dir" is given a value that is the full pathname of the directory represented by "../../schema" in the qtspecs tree. On one editor's local system, it might have the actual value "E:\w3ccvs\WWW\XML\Group\qtspecs\schema", while another might have the value "/usr/w3c/XML/Group/qtspecs/schema".

Those uses of "../" are, of course, the way of representing the idea "look in the parent directory of the directory currently in focus". In this case, "the [initial] directory currently in focus" is the directory in which the particular build.xml file for a given document is located.

2.1.4 import

Ant scripts are able to import XML files using ant script syntax. Such imports might set the values of properties, define common targets, *etc*.

2.2 Specific build information

2.2.1 Tasks When Creating A New Document

Once in a while, QT creates a new document. For example, the XML Query WG recently created a new document for XQuery 1.1. Whenever a new document is created, there are several things that must be done to ensure that the new document is "recognized" in the context of the build system.

1) The file …/qtspecs/schema/xsl-query.dtd contains an entity declaration:
   ```
   <!ENTITY % specAbbreviations…
   ```
   That entity is declared to be a list of abbreviations, one for each document known to the build system. For example, the symbol associated with XQuery 1.0 is "XQ", while the symbol associated with XQuery 1.1 is "XQ11".

2) The build.xml file that builds each document (*e.g.*, …/qtspecs/specifications/xquery11/build.xml) contains a property definition (see 0 above):
   ```
   <property name="spec.code" value="…
   ```
   That property value must contain one of the values that are included in the entity specAbbreviations (see list item 1) above).

3) The file …/qtspecs/style/xsl-query.xsl must be edited to add support for the new document. In this stylesheet, there is a variable definition for a variable named "default.specdoc". The value of that variable is defined using an <xsl:choose> element. Each <xsl:when> child of that <xsl:choose>

performs a test to determine whether the document being compiled is the document for which the <xsl:when> is intended. If so, the variable default.specdoc is assigned a "code" associated with that document. (For example, "DM" is the code associated with the XQuery 1.0 and XPath 2.0 Data Model document, and "XQX11" is the code associated with the XQueryX 1.1 document.) In addition, an <xsl:message> element generates a console message identifying the document being compiled (by displaying the value of the code).

When each document is (successfully) "compiled", several files are created. Obviously, one of those files is the HTML file containing the result of the build process. Not so obviously, the act of compiling the document also produces a file called an "anchor document".

Anchor documents are XML files that appear in the …/qtspecs/etc/ directory. They make it possible to symbolically reference some location in one of our documents from another document (*e.g.*, to reference a particular HTML <h1> in the XQuery 1.0 document from the XQueryX document). For example, the following lines might appear in an anchor document:

```
<div1 id="intro">
<head>
1 Introduction</head>
</div1>
<termdef xmlns:e=http://www.w3.org/1999/XSL/Spec/ElementSyntax
    id="dt-sequence" term="sequence"/>
```

The filenames of anchor documents are of the form "XX.xml", where "XX" is the value of the spec.code property (see list item 2) above) defined in the build.xml file for the corresponding "real" document. Thus, the anchor document corresponding to the XQuery 1.1 spec is named "XQ11.xml".

2.2.2 Entity Declarations in Documents

The first lines of the "root" file of each of our documents almost always contain an internal DTD subset. For example, the source file for XQueryX 1.0 are:

```
<!DOCTYPE spec SYSTEM "../schema/xsl-query.dtd" [
<!ENTITY date.day "23">
<!ENTITY date.DD "23">
<!ENTITY date.month "January">
<!ENTITY date.monthnum "01">
<!ENTITY date.year "2007">
<!ENTITY doc.date "&date.year;&date.monthnum;&date.DD;">

<!ENTITY w3c.tr "http://www.w3.org/TR">
<!ENTITY doc.shortname "xqueryx">
<!ENTITY doc.w3c-doctype "rec">
<!ENTITY doc.w3c-designation "REC-&doc.shortname;">
<!ENTITY doc.w3c-prev-designation "PR-&doc.shortname;">
<!ENTITY doc.public
    "&w3c.tr;/&date.year;/&doc.w3c-designation;-&doc.date;">
<!ENTITY doc.latestloc "&w3c.tr;/&doc.shortname;">

<!ENTITY xqx-schema SYSTEM "included-files/xqueryx-CDATA.xsd.xml">
<!ENTITY xqx2xq-stylesheet
    SYSTEM "included-files/xqueryx-CDATA.xsl.xml">
<!ENTITY xq-grammarSnippet SYSTEM
    "included-files/xqueryx-grammar.xml.snippet-CDATA.xml">
<!ENTITY xqx-schemaSnippet SYSTEM
    "included-files/xqueryx.xsd.snippet-CDATA.xml">
<!ENTITY exmpl1xq SYSTEM "included-files/exmpl1-CDATA.xq.xml">
<!ENTITY exmpl2xq SYSTEM "included-files/exmpl2-CDATA.xq.xml">
<!ENTITY exmpl3xq SYSTEM "included-files/exmpl3-CDATA.xq.xml">
```

```
<!ENTITY exmpl1grammar SYSTEM
    "included-files/exmpl1-CDATA.grammar.xml">
<!ENTITY exmpl4xq SYSTEM "included-files/exmpl4-CDATA.xq.xml">
<!ENTITY exmpl1xqx SYSTEM "included-files/exmpl1-CDATA.xqx.xml">
<!ENTITY exmpl2xqx SYSTEM "included-files/exmpl2-CDATA.xqx.xml">
<!ENTITY exmpl3xqx SYSTEM "included-files/exmpl3-CDATA.xqx.xml">
<!ENTITY exmpl4xqx SYSTEM "included-files/exmpl4-CDATA.xqx.xml">
<!ENTITY exmpl1xqx2xq SYSTEM
    "included-files/exmpl1-CDATA.xqx2xq.xml">
<!ENTITY exmpl2xqx2xq SYSTEM
    "included-files/exmpl2-CDATA.xqx2xq.xml">
<!ENTITY exmpl3xqx2xq SYSTEM
    "included-files/exmpl3-CDATA.xqx2xq.xml">
<!ENTITY exmpl4xqx2xq SYSTEM
    "included-files/exmpl4-CDATA.xqx2xq.xml">
<!ENTITY mimetypeappendix SYSTEM "xqueryx-mime-type.xml">


<!ENTITY % status-entities SYSTEM
    "../etc/status-entities.dtd">
%status-entities;

<!ENTITY status-section-id "id-status">
<!ENTITY spec-devby "&xqueryx-devby;">
<!ENTITY changelog-id "&xqueryx-changelog-id;">
<!ENTITY changes-para "&post.PR.changes;">
<!ENTITY implementation-report-location "&xquery-impl-report;">
<!ENTITY implementation-report-availability "&report-public;">
<!ENTITY implementation-para "&implementation-report-exists;">
<!ENTITY disclosure.one "&disclosure.xquery;">
<!ENTITY Bugzilla-key "[&xqueryx-Bugzilla-key;]">
<!ENTITY patent-policy-paragraph "&ppp-one;">

<!ENTITY status-section SYSTEM "../etc/status-REC.xml">
]>
```

The lines from `<!DOCTYPE` through `<!ENTITY doc.latestloc` are central to the creation of each of our documents. At the time of writing, there was little common practice regarding the specific entities that are declared and how they are used in each document.

Some time in 2008, that will be rectified throughout the …/qtspecs/… tree.

2.3 Stylesheet Relationships

A number of XSLT stylesheets are used in the qtspecs build system. Many of those stylesheets are located in the shared styles directory, while others are found in the local styles directory for specific documents. In the …qtspecs/specifications/grammar-10/ and…qtspecs/specifications/grammar-11/ directories, some stylesheets are located in the …/parser/ and …/lalr/ subdirectories.

There is a somewhat complex and intimidating relationship between the many stylesheets found in the qtspecs build system (approximately 110 stylesheets in all). Understanding those relationships is important in understanding how the qtspecs build system works, yet difficult to do without some guidance.

There are at least four different sets of relationships, each corresponding to a category of activity performed by the ant script (build.xml) for a specific document.

2.3.1 Creating document-specific grammar files

Some, but not all, QT specifications involve the specification of language grammar. This activity does not exist for specifications that do not (as well as requirements and use cases documents, which never do).

Typically (except for XQuery and XPath themselves), two temporary grammar files are created: temp-xquery-grammar.xml and temp-xpath-grammar.xml. XQuery requires only temp-xquery-grammar.xml and XPath requires only temp-xpath-grammar.xml.

The only stylesheet involved in producing the temporary grammar files is named “strip.xsl”. That stylesheet is found in the …/qtspecs/specifications/grammar-XX/parser/ directory (where “XX” is either “10” or “11”, depending on the specific spec being built).

The “relationships” of stylesheets used to create document-specific grammar files is illustrated by:

```
strip.xsl
```

NOTE: There is also a stylesheet in the shared styles directory named “strip.xsl”, the contents of which contents are slightly different from the contents of the strip.xsl stylesheets found in the grammar parser directories. The only place I have been able to find where this version is invoked is in the build.xml files of the grammar-11 directory itself. It is apparently used only during the creation of temporary grammar files during production of xpqlexnote.html and ebnf.html.

2.3.2 Constructing an “assembled” XML source file

Many QT specifications are built using a number of source files. A simple identity transform of the root file of many of those specifications is sufficient to produce a single XML file containing all lines of source code. The stylesheet used to perform this transformation is cleverly named “identity.xsl” and is found in the shared styles directory.

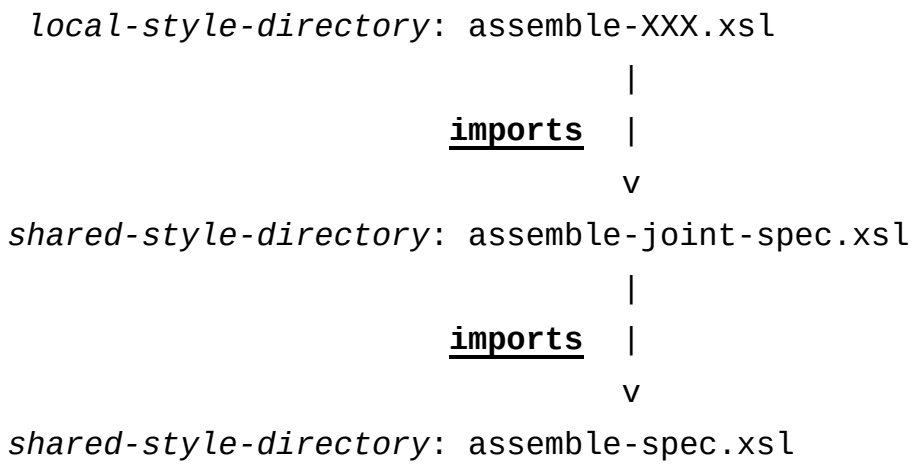
However, specifications that incorporate grammar must be *assembled* with considerably more complex transformations. In particular, the grammar defined in the temporary grammar files is not in a form suitable for direct transformation into an HTML specification, but must be transformed in several ways. Perhaps the most important, and complex, such transformation is generation of a “hot-linked” BNF nonterminal symbol everywhere that nonterminal is used in the specification (except, of course, for its definition).

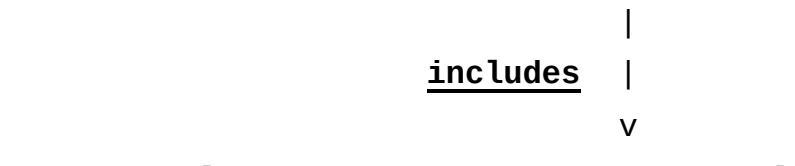
NOTE: The various requirements and use cases documents do not undergo construction of an assembled XML source file. Because none of them incorporate grammar, only an identity transform would be required in any case; however, because they are all composed of a single XML source file, even the identity transform is not required.

The “relationships” of stylesheets used to create assembled XML source files of documents that do *not* incorporate grammar is illustrated by:

```
identity.xsl
```

The relationships of stylesheets used to create assembled XML source files of documents that *do* incorporate grammar is illustrated by:



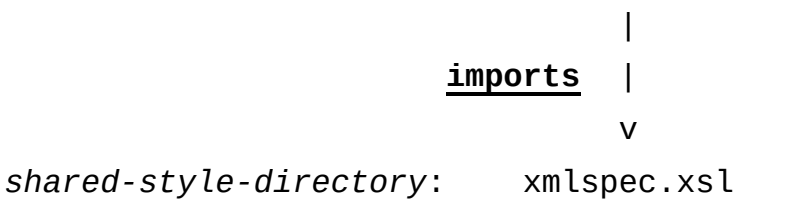


where “XXX” is “full-text”, “update”, “xquery”, “xqueryx”, *etc.*

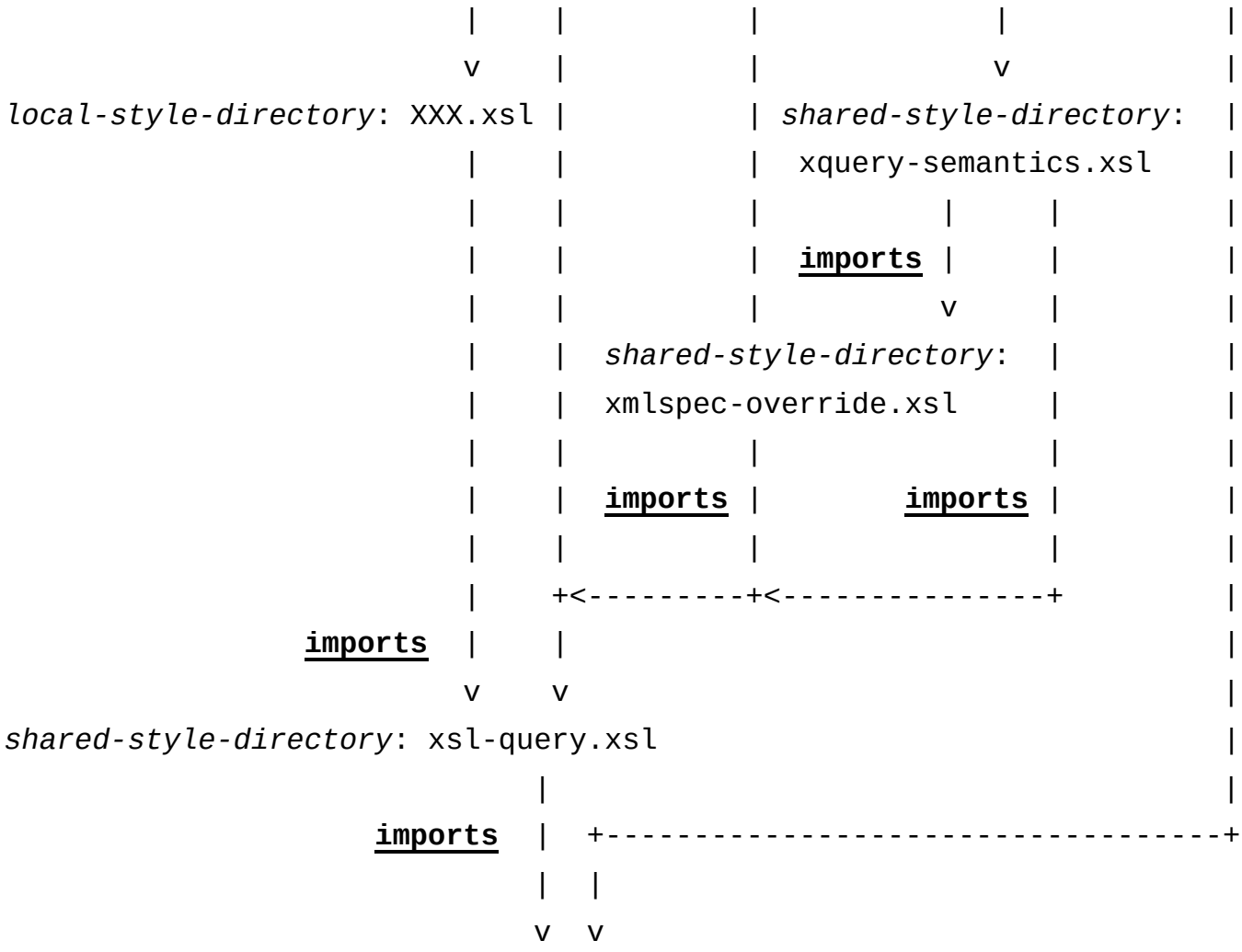
2.3.3 Generating the HTML version of a document

Once an assembled XML source file has been generated (or not, for requirements and use cases documents), the HTML version of the document is produced. The process differs slightly for specifications *versus* requirements and use cases documents.

Requirements and use cases documents are built using stylesheets having the following relationships:



By contrast, specification documents are built using stylesheets having the following relationships:



shared-style-directory: xmlspec.xsl

That set of relationships certainly looks complex. Here's a summary of *why* the relationships are what they are:

- Some documents (at time of this writing, only Full Text) have their own local stylesheets that import xsl-query.xsl that are directly invoked by the ant xslt task.
- Some documents (those, other than Full Text, that incorporate grammar) are built by an ant xslt task that invokes xquery-semantics.xsl. That stylesheet imports both xmlspec-override.xsl *and* xsl-query.xsl.
- Some documents are built by an ant xslt task that invokes xmlspec-override.xsl, which in turn imports xsl-query.xsl.
- Some documents are built by an ant xslt task that invokes xsl-query.xsl directly, which imports xmlspec.xsl.
- Finally, some documents are built by an ant xslt task that invokes xmlspec.xsl directly.

NOTE: Other than the fact that documents that incorporate grammar either have their own local stylesheets or are build by ant tasks that invoke xquery-semantics.xsl, I have not yet decoded the conditions that cause some documents to require xmlspec-override and some to not require that stylesheet. The same is true of xsl-query.xsl. Editors who understand the reasons are encouraged to inform the rest of us!

2.3.4 Producing the anchor document for a document

Each QT specification (and, soon, each requirements and use cases document) have an associated anchor document that is created just after the document itself.

The anchor documents are built using stylesheets having the following relationships:

2.3.5 Other stylesheet-using tasks

There are several other stylesheets that are used in the production of one or more specifications, requirements, or use cases documents. Their relationships are generally easy to discern simply by reading the associated build.xml files and are thus not documented here. If there are requests to document them, then a future version of this document will do so.

TO BE DONE:

- Specify the first few lines of each spec using entities, etc. and explain why
- Explain structure of build.xml files, especially use of property declarations
- DIFFERENT EFFORT: Document the DTDs/Stylesheets that are used in the QT specs.
- DIFFERENT EFFORT: Normalize all build.xml files in several ways: define and use same properties, have all the same targets (e.g., clean, all, spec, stage, etc.)
- DIFFERENT EFFORT: Normalize all sourcefile.xml internal subset DTD