Guide to Customizing Templates

for Virage Solution Server 4.0

Virage customers may make a reasonable number of print or electronic copies of this document for internal use only. This document can change without notice. Virage, Inc. assumes no liability for any errors it might contain.

Virage, Inc., San Mateo, CA 94402

© 2002 Virage, Inc.
All rights reserved. Published 2002
Printed in the United States of America

1003211 March 15, 2002

"Virage," "PinPoint," "VideoLogger" and "AudioLogger" are registered trademarks of Virage, Inc. The Virage logo, "MediaSync," "MyLogger," "SmartEncode," "Virage ControlCenter," "AudioLogger," "Virage Solution Server," "Video Application Server," "Internet Video Guide," and "VIR Image Engine" are trademarks of Virage, Inc.

All trademarks are the property of their respective owners.

The following notices apply to certain Virage products:

Visual Information Retrieval technology, used in certain Virage products, is covered by U.S. patent numbers 5,893,095, 5,911,139, 5,913,205, 5,915,250, 5,983,237, and 6,084, 595.

Operation of certain Virage products is covered by U.S. Patent 5,579,471 licensed to Virage by IBM Corporation.

Certain Virage products incorporate International Components for Unicode (ICU) software, Copyright © 2000, International Business Machines Corporation and others. All Rights Reserved. The source code for ICU may be obtained via the Internet at http://oss.software.ibm.com/icu.

A portion of Virage audio products contains ViaVoice technologies for Broadcast Speech Transcription and Speaker Identification licensed to Virage by IBM Corporation.

A portion of Virage audio products contains audio-classification technology licensed to Virage by Muscle Fish LLC.

The Virage Media Analysis Plug-in for Face Recognition contains face detection and recognition technology licensed to Virage by Visionics Corporation®.

The Virage Media Analysis Plug-in for On-Screen Text Recognition contains text extraction technology licensed to Virage by SRI International®.

Parts of the JPEG image read-write software is copyright 1991–95 by Thomas G. Lane. Actions of Virage products that read or write JPEG files are based in part on the work of the independent JPEG group.

Actions of Virage products that read or write MPEG files have been derived from Berkeley MPEG software.

Virage distributes QuickTime software under license with certain Virage products.

ActiveState, ActivePerl, and PerlScript are trademarks of ActiveState Tool Corp.

The Video Application Server product includes software developed by the Apache Group for use in the Apache HTTP server project (http://www.apache.org/).

Mozilla Public License for Expat, Version 1.1: The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http:/www.mozilla.org/MPL/. Software distributed under the License is distributed "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is expat version 1.1. The Original Developer of the Original Code is James Clark. Copyright (c) 1998, 1999. All Rights Reserved.

Contents

Preface ix

1 Introduction to Templates 14
- The sample views 15
- Location of templates and related files 16
- JavaScript and SMIL tools that the sample templates use 18
- Virage attribution 20
- Further customizing 20

2 Customizing the Templates 22
- VDF metadata: data types the index contains 22
- How the server uses templates and processes variables and queries 26
- Constraining a query with hidden HTML form variables 27
- Search and results templates 28
- What do search and results templates look like? 31
  - Example: template with embedded variables and VTL statements 31
  - Example: results template that displays multiple results 32
- Variable substitution and example result hash 34
- Requesting information to display 36
  - Which query control parameters to use 36
  - Query control examples 37
- Where to set variables 42
- Variable parsing sequence 46
- Sample tasks for customizing templates 49
  - Creating new templates 49
  - Specifying the template to render 49
  - Modifying background colors 50
  - Modifying the size of the embedded player 51
  - Displaying multiple proxy links on a results page 52
  - Displaying additional text tracks 52
  - Searching by dates and date ranges 53

Sorting by a specified field 54
Modifying the proxy bitrate and format chooser 55
Customizing playlist functionality 57
The playlist Perl module and CGI 57
Invoking the playlist in the templates 58
Adding an interstitial to a playlist 62
Developing templates with folders 63
Folders and the user.pm module 65
Using the JavaScript library 66
Working with international templates 69
Set character encoding to native codepage. 69
Avoid font-family specifications 70
Compatibility with pre-release 2.2 templates 70

3 Using the Virage Template Language 72
Variable syntax 73
Using variable modifiers 73
Using VTL_ tokens 74
Macro substitution 75
Looping variables with VTL_LOOP 77
Including template files: VTL_INCLUDE 78
Using conditional statements: VTL_IF, VTL_ELSE, and VTL_ELSE_IF 79
Accessing a Perl module from a template file: VTL_SCRIPT . . . 80
Calling user functions within a template: VTL_CALL 81
Troubleshooting and tracking to a log file: VTL_LOG 82
Perl access functions 83
User functions 83
Perl function example. 84
System variables. 85
Search variables 86
Result variables 97
Result arrays 104
Error handling 110
Internationalization issues 110
Executing vss_SR from the command-line 111

A The URI Builder Script 112
URI structure 113
Semicolons or ampersands? 115
The problem 117
A solution 118

B SMILClock 122
Synopsis 122
Description 122
clock value 122
clip-begin 123
Examples 128
Files 128
See also 128
Caveats 128
Bugs 129

C Template File Dependencies 130

D Template Site Map 140

Glossary 154

Index 158

Illustrations

1–1 Directories pertinent to template design 17
2–1 VDF metadata structure and time-based track information 26
2–2 Template-CGI workflow within the solution server 30
2–3 Variable parsing order 47
2–4 Process for invoking templates 48

Preface

Welcome to the *Guide to Customizing Templates for Virage Solution Server*. This guide describes how to customize the Virage Solution Server user-interface look and feel as well as functionality. The Virage Solution Server templates are HTML templates with an embedded Virage Template Language that you can use to interact with asset metadata.

Web developers can use this guide to modify the templates that ship out-of-the-box, to add additional web pages to the application, or create a new web interface to interact with the core solution server.

Accompanying this guide is an online VTL Tutorial that guides you through Virage Template Language examples and an Advanced Search view you can use as a basis for developing complex queries. To view the VTL Tutorial or the Advanced Search view, create an account with the appropriate views within the application user-interface Account Manager once you install the Virage Solution Server SDK.

Audience

The audience for this guide is web masters assigned the to set up and customize the Virage Solution Server interface look and feel as well as the base functionality. This guide is written for web developers with knowledge in advanced HTML, JavaScript, and Perl.

Related documents

The following documents describe related concepts or tasks:

- The `readme.htm` file describes any known issues and bugs and lists the new features for the release.

- *The Virage Solution Server Getting Started Guide* ships with the base product and describes installation and migration procedures, deployment considerations, and troubleshooting. The guide includes information on setting up security with LDAP and a database for storing folder data.
- The Virage Solution Server online help describes options you can perform in the interface. The help is context sensitive and pertinent to the current page.
- The *Virage Solution Server Administrator's Guide* ships with the base product describes tasks administrators perform. This includes how to use the XML configuration files and how to perform tasks in the user-interface or on the command line. The guide provides a conceptual overview of the application architecture and the administrative tasks of managing users, views, accounts, indexes of assets, and user folders. The guide also describes the installation directory structure of the solution server.
- The *Virage Solution Server SDK Programmer's Reference* describes tools developers can use to further customize and integrate the solution server with existing systems and how to integrate user authentication security within a custom extension to the application. The programmer's reference also maps all the application CGIs to each page within the interface along with the corresponding CGI actions and permissions that apply to each template file.
- The online VTL tutorial and Advanced Search view provide examples and sample code you can use to develop advanced templates. To use the tutorial or Advanced Search view, install the SDK and create an account in the Account Manager administrative web pages with the tutorial or Advanced Search views.

Using this guide

This document describes how to use Virage Solution Server.

Chapter 1 introduces the solution server architecture.

Chapter 2 explains how to customize the templates.

Chapter 3 describes the syntax of the Virage Template Language and provides basic examples.

Appendix A explains the JavaScript URI builder tool used throughout the sample templates.

Appendix B explains the SMILClock tool used in the sample templates.

Appendix C describes which template files affect other pages within the sample views.

Appendix D includes a site map of the sample templates.

The Glossary defines some commonly used terms.

How to obtain most current information

For the latest information, between product releases, about issues that affect the product, visit the Virage Knowledge Base at the following location:

http://www.virage.com/products/support/knowledgebase/

In addition, links to all Virage product documentation are available on the Virage Support website at the following location:

http://www.virage.com/products/support/documentation/

Log in as `virage` with a password of `virtu`. To check if you have the latest version of the documentation, compare print dates and versions on the copyright page.

Contacting Virage

Send questions about purchasing the Virage Solution Server product to

info@virage.com

Contact our technical support staff if you have questions about using Virage Solution Server or suggestions for making it better. Call 650/372-2645 between 7:00 A.M. and 4:00 P.M. Pacific standard time, Monday through Friday, or send email to

support@virage.com

Some of your questions might be answered on our website. You can access our site at the following location:

http://www.virage.com/

Support for ActivePerl

Commercial support for ActivePerl is available through the PerlClinic at http://www.PerlClinic.com. Peer support resources for ActivePerl issues can be found at the ActiveState website under support at the following location:

http://www.activestate.com/support/

1 Introduction to Templates

Virage Solution Server is a server side web application that provides an HTML-based interface to search, share, and store archives of indexed assets. Virage Solution Server is implemented with CGI (Common Gateway Interface) programs. When a user submits a query through the application interface, the CGI programs access XML configuration files to determine information, such as the location of pertinent files, and use HTML templates to format results and return them to a web browser.

This guide reviews the web developer's role and capabilities to customize the application look and feel and functionality. This guide describes how to customize the templates beyond simple modifications in the XML configuration files on in the interface. This includes customizing Perl and using the Virage Template Language that is embedded within the HTML files.

As a web developer, you can modify the default interface or generate your own interface, by editing the following items:

- HTML templates
- embedded Virage Template Language tags within the HTML templates
- JavaScript files
- XML configuration files
- a Perl module, `user.pm`, for each account
- Perl CGI modules

This chapter introduces the sample views that ships with the product, the location of files pertinent to template design and development, and introduces the basic tools used in the sample templates.

The sample views

The application ships with sample templates, or views. A *view* is defined as a set of distinct HTML templates that determine the look and feel as well as the application logic that the end user experiences.

The solution server base product ships with three sample views—a Search view, an Edit view, and a Folders view. The Virage Solution Server SDK ships with additional views—a Virage Template Language Tutorial view, an Advanced Search view, and a sample Export plug-in.

The sample Search view is for the end user to search through archives of assets and the Edit view contains links to allow for modifying metadata that describes the assets. The Folders view allows end users to gather assets into a playlist and to play these sequentially (You can only play assets that play, such as videos or video clips). Users can also share folder contents with others. Table 1–1 reviews the purpose of the sample views.

Table 1–1: Sample default views

View	Purpose
Edit	Add or delete assets to the index with which the account is associated. Search for and modify metadata.
Search	Search the index and view search results
Folders	Collect assets into folders. You can play clips or videos sequentially and share folder contents with others
Tutorial	Learn the Virage Template Language through a series of annotated search and result templates and lessons
Advanced Search	View samples of complex queries. You can use the template to perform complex queries and view the XML commands in the XML log.
Export plug-in	Use the sample plug-in to export a playlist into one of several formats

The *Virage Solution Server Administrator's Guide* further describes the base sample views and view management, such as how to create and delete views.

Location of templates and related files

There are several directories pertinent to the template designer. These include the directory location to the parsed template files, static (non-parsed) content (such as GIF images or cascading style sheets), a Perl module (`user.pm`), and the master templates that are copied to create all new templates.

Figure 1–1 illustrates the location of directory files pertinent to the template designer. The template files are located in the subdirectory for each view within the `conf/` directory relative to the root Virage Solution Server installation. Most of the sample template files have the `.tmpl` extension. However, this extension is not necessary. The `.tmpl` indicates that these files contain the Virage Template Language code within the file. However, you can use any HTML or text file, for example, with the solution server. For example, the sample template files are located in this location: `VS\conf\accounts\`*my_account*`\`*my_view*`\`*my_template.tmpl*.

Non-parsed (static) content for each account is located within the `vssdocs/accounts/` directory relative to the root solution server installation. For each account, you can extend functionality using a Perl module called `user.pm` that exists for each account. There is a `user.pm` file in each subdirectory for each account in the `VSS_Accounts` directory. For example, for the sample account, the `user.pm` file is located in `VS\lib\VSS_Accounts\sample\user.pm`. The subroutines in the `user.pm` module are called every time a page is rendered in the account. (For more information on the `user.pm` file, see "Perl access functions" on page 83.) The `user.pm` file is located within the `lib\VSS_Accounts\sample` subdirectory.

Within the `exten\` directory, which contains application extension files, there is the `vssext\` subdirectory that contains the sample application extension shipped with the product. The `sample` subdirectory contains the `master_templates\` subdirectory that contains the master template files that the solution server uses to create new accounts and views. Within the `master_templates\` subdirectory there are additional subdirectories: `sample_acct\` and `views\`. The `sample_acct\` directory contains files that

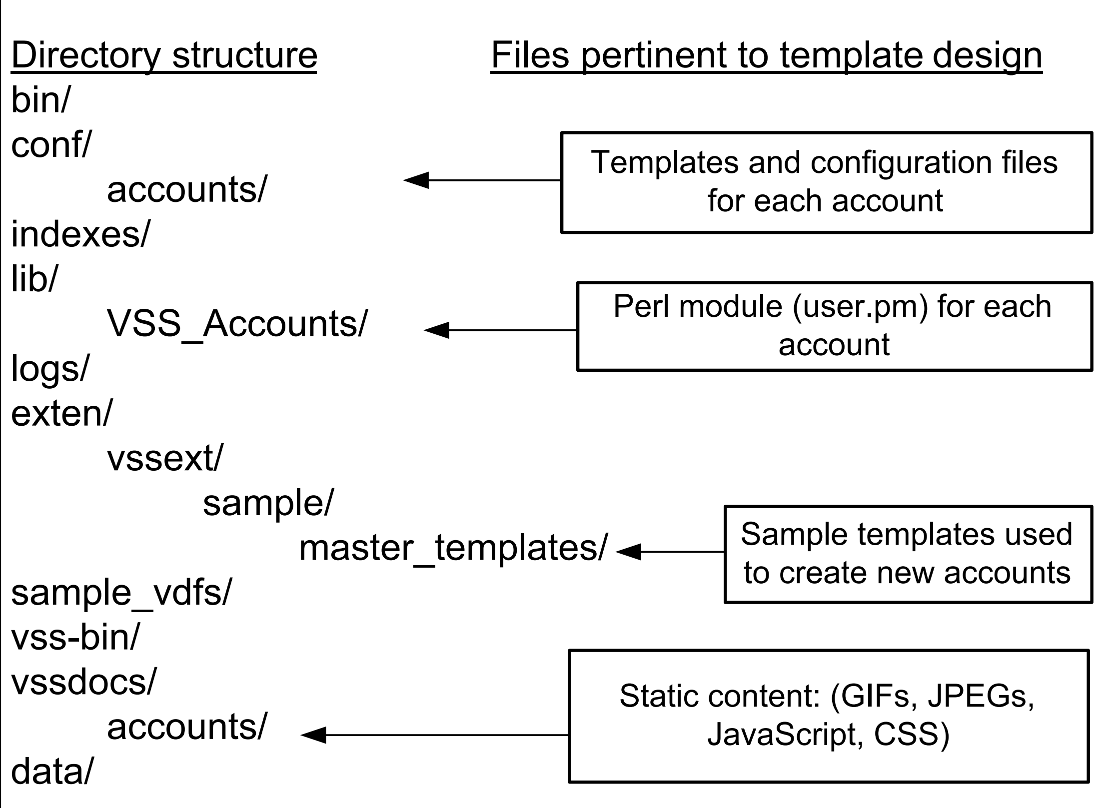


Figure 1–1: Directories pertinent to template design

the solution server copies to create each new account. The `views` directory contains templates and static files that implement a particular sample view, such as a Folders view or an Edit view.

All the master templates and static files, except for the default Administration web pages templates, reside in the `vs/exten/vssext/sample/master_templates/sample_account/views` subdirectory. The templates for the Administration pages reside in the `conf/admin/admin_view` directory.

If an administrator uses the Account Manager web pages to create a new account, he or she can choose to create that account with views based on existing views. If the administrator creates the new account with an existing view, the solution server copies master templates from the

`master_templates` subdirectory and copies them (along with the configuration files) into the `conf/` directory, which is further subdivided by each account and view. You can customize the views within the new location to suit your content.

For example, if you create an account called `Videos From Mars` in a directory called `space_videos` and specify a view based on the sample Edit view, the solution server copies the sample Edit view templates into the following directory: *INSTALL_ROOT*`/VS/conf/accounts/space_videos/edit/`.

Note: Changes to the master templates do not affect any current views, but changes to the master templates do affect all views created subsequently.

JavaScript and SMIL tools that the sample templates use

The sample templates employ tools you need to know about in order to customize the sample templates. These are the URI Builder JavaScript tool and the SMIL Clock tool.

The URI Builder JavaScript tool builds links to new search pages and facilitates typing long URLs in templates. (A URI has a specific formal definition in web programming; however, in this guide a URI is the same as a URL. We use URI to refer to code that uses this acronym and URL otherwise.) The SMILClock tool helps to convert between various time formats.

Note: As you develop new templates, you can choose not to use the URL Builder JavaScript tool. For example, some HTML validators (and browsers) prefer that contents within an HTML `href` tag do not contain any new line characters, which the URI Builder tool introduces. The browsers that VSS supports do, however, support these characters.

In brief, without the URI builder, a URL in the templates looks like this (without the line breaks this example contains):

```
<a href="${nav_search_uri}/${account}/${view}?tem-
plate=viewer_frameset.tmpl&amp;proxy_mime_type=${prox_mime_typ
e}&amp;search_mode=${search_mode}&amp;result_type=${result_typ
e}&amp;asset_id=${asset_id}&amp;video_asset_id=${video_asset_i
d}&amp;query=${query:j:h}<VTL_IF
NAME="result_type"VALUE="clips">&amp;timein_msec=${cinfo_timei
n_msec}&amp;timeout_msec=${cinfo_timeout_msec}&amp;timein=${ci
nfo_timein}&amp;timeout=${cinfo_timeout}</VTL_IF>">
```

With the URI Builder, you can format the same information in a form that is easier to read and the URI Builder tool converts this to a machine readable URL. This is the format the same templates use to invoke the URI Builder.

```
<a href="javascript: location = URI_builder(
'SCHEME',
'AUTHORITY',
'PATH',
'${action_view_uri}',
'${account}',
'${view}',
'QUERY',
'template=viewer_frameset.tmpl',
'proxy_mime_type=${proxy_mime_type}',
'search_mode=${search_mode}',
'result_type=${result_type}',
'asset_id=${asset_id}',
'video_asset_id=${video_asset_id}',
'query=${query:j:h}',
<VTL_IF NAME="result_type"
VALUE="clips">
'timein_msec=${cinfo_timein_msec}',
```

```
'timeout_msec=${cinfo_timeout_msec}',
'timein=${cinfo_timein}',
'timeout=${cinfo_timeout}',
</VTL_IF>
'FRAGMENT'
)">
```

The JavaScript URI Builder tool is described in detail in Appendix A "The URI Builder Script" and the SMILClock tool is described in Appendix B "SMILClock."

Virage attribution

Applications developed with Virage Solution Server must include proper attribution to Virage. This includes placement of the appropriate Virage icon within each search and results web page, as illustrated in the sample search interface.

Further customizing

This guide outlines the customizing that a web developer can perform. A developer or programmer can further customize the functionality of the solution server, such as to add custom modules to the application. If your application requires integration with other systems, work-flows, or data sources, you might need to perform code-level integration. For this purpose, Virage publishes the *Virage Solution Server SDK Programmer's Reference*.

2 Customizing the Templates

This chapter discusses how you can develop web pages using the Virage Solution Server architecture. This chapter reviews conceptual information about how the solution server processes templates as well as techniques for incorporating assets into the Virage Solution Server web pages.

The end user experience—the look and feel as well as much of the functionality of the solution server views—is determined within the template files. Most of the content that an end user sees is generated on the fly from a combination of templates along with the results of a query. Virage Solution Server includes powerful tools for the web designer to create and customize templates to meet varying content needs. These tools include a powerful embedded template language, the Virage Template Language (VTL), designed to facilitate efficient searching and dynamic rendering of results, as well as the ability to extend the functionality with Perl and JavaScript.

As a web developer, you can use these tools to integrate with existing content, tailor the look and feel, or incorporate unique branding. As a web developer, you can create new templates or customize the sample templates that are shipped as part of Virage Solution Server. For designers who are modifying the sample templates (rather than creating your own), this chapter also includes examples to make common modifications.

VDF metadata: data types the index contains

When a CGI submits a search to the search engine, the search engine queries an index of files uploaded to the solution server. The uploading process is described in the overview of the *Virage Solution Server Administrator's Guide*. This section, however, reviews the internal structure of a VDF file, so that as a template designer, you understand the data structure, including

the data types, that the search engine queries, and in some cases modifies, within each VDF file.

Note: You can upload into the VSS index many different document types (described in the *Virage Solution Server Administrator's Guide*). Once you upload any asset into the VSS index, VSS encloses the asset in a VDF file. For example, if you upload an Adobe PDF document into VSS, VSS creates a VDF file and stores the PDF content within the VDF binary track. (Of course, videos and video clips logged in the VideoLogger application originate as VDF files.)

For example, if the CGI queries for clips labeled "Home Run," the CGI searches the VDF files for a field within each VDF structure called `clabel_name` where the value equals "Home Run." To review, as a template designer, you access the individual VDF fields with the Virage Template Language within the templates and the CGI sends the request to the underlying server.

Note: The definition of a field in the solution server has a broader scope than the definition of a field in the VDF file. The solution server field includes any portion of the VDF file that you can search, including text track records or clip label records, for example.

The VDF metadata is organized into time-based data tracks. Figure 2–1 illustrates the VDF metadata structure. The VDF file organizes all the information that pertains to the source video, such as when the file was created, and proxy format, or pixel height, width, duration, or proxy offset of the clip or video.

For example, you can modify the proxy offset to control the timing of when clips play. The proxy offset value stored in the VDF is the number of milliseconds between the start of the proxy encoding and the start of video metadata capture. The offset value can be positive or negative. A positive

offset means that the proxy video starts later than the start of the metadata capture. If you customize the sample video viewer or create your own stand-alone player, you can access the VDF proxy offset and modify the offset value in a template to control the exact time a clip begins to play.

Each field within a VDF file has a data type and an input type. The data type determines whether the field contains strings, dates, or numbers. As you construct queries, use appropriate operands per the data type. For example, the VDF creation date is a date field. You can construct a query based on when the clip or video was created with the VIR_VDF_CREATION_DATE to search for only assets created on a specific date. Or, if you search with a wider scope for more VDF files, you can control the display in a results template by displaying the vinfo_creation_date result variable. If you construct a query that seeks a date range, you can use an operand, such as `eq`, `gt`, `lt`, `geq`, or `leq`. (The operands discussed in "Using query operands" on page 93.

The input type determines whether the VDF field holds a single value, multiple possible values, or a single value from a select list of choices. In other words, individual field values are either simple single fields or an item list. You can access the values in a VDF files and render multiple values in the user-interface as pull-downs or check boxes, for example, if the field contains an item list.

In every case, you can access a result that is one or more selections from a fixed list of choices within the VDF file. Each VDF track contains information about a specific index to the source asset: closed-caption text, keyframes, if any, and so on. The VDF header structures information describes the contents of the metadata, including which tracks contain data and how many records are in each populated track. The components of the VDF file are further detailed in the *VideoLogger SDK Programmer's Reference*.

In addition, the solution server adds some additional header elements to the index that are not present in the VDF file. These include information such as the date the VDF was added to the index, the name of the VDF file, and asset ID numbers for each video and clip label in the VDF. VSS also adds information to VDF files it creates for document types that originated as other media types (not VDF files). For example, if you upload a Microsoft PowerPoint document to the index, VSS creates a VDF and adds information to describe the asset. You can include these fields as constraints in a search and display these values to the user in template files.

When you upload an asset in the user-interface, you can specify certain defining characteristics of that asset, such as a title, author, and keywords. The values you enter in the interface override existing values. For example, if you upload a Microsoft Word document that has the author property defined as Bill and you specify a different author name in the VSS upload process, the new name is stored in the VDF file.

Within the sample Edit view, users can modify several fields (such as in- and out-times or the duration of a video clip) and this modifies the data in the VDF file. From the sample Folders view, however, users can modify clips; however the VDF is not modified. The user changes are only stored in a separate database for Folders data if you specify to save the Folder information to the database server. The sample Search view, of course, is read-only and does not provide an interface for users to modify the metadata.

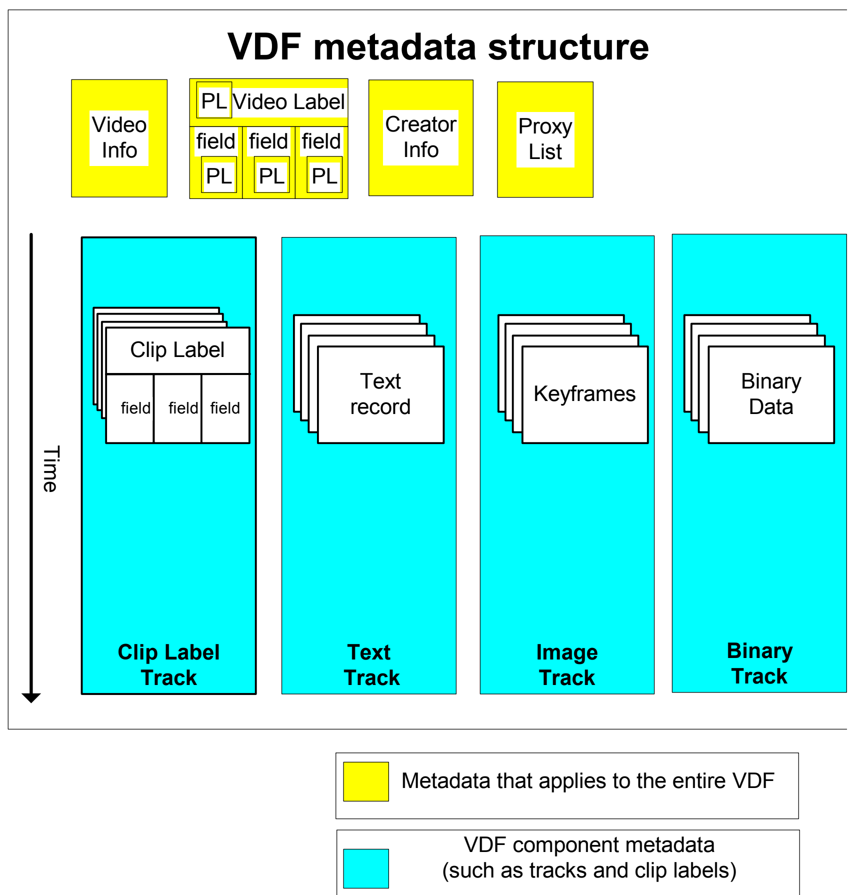


Figure 2–1: VDF metadata structure and time-based track information

How the server uses templates and processes variables and queries

This section describes how Virage Solution Server processes template files, so you can determine which files to customize to meet your needs. As a web developer, you create templates that contain variable macros. A template provides the format to display data to a web browser. A template is comprised of HTML (and JavaScript) with special embedded Virage Template Language tags that include variables, variable modifiers, and tokens. Fundamentally, the Virage Template Language is simple to use.

When Virage Solution Server queries for data, the templates, along with the variable macros, dictate what is queried to the underlying index as well as what is displayed and how it is formatted to the client browser.

There are two types of Virage Template Language variables: scalars and arrays. Scalar variables are represented in the templates with the following syntax: `${`*variable_name*`}`. Arrays are accessed with a `VTL_LOOP` token. The CGI loops through the array to access multiple values. (For an example of what an actual template looks like that contains this syntax, see “What do search and results templates look like?” on page 31.)

When the main Virage Solution Server CGI, (`vss_SR` on Solaris or `vss_SR.exe` on Windows), processes a request, it creates a list of the variables from various locations and loads a template. The CGI fills in, or resolves, the values for the variables (such as an actual name of an account for the variable `account_name`) and uses another template to present the results to the user in a browser.

Constraining a query with hidden HTML form variables

You can set variables within your templates by using HTML forms. You can use a combination of user-visible form controls and hidden form variables to build and customize a query. For example, you can compose a search whereby you expose certain choices in an HTML pull-down box and use hidden form variables to constrain the search. For example:

```
<form name="searchform" action="${nav_search_uri}/${account}/${view}"
method="get">
Search the ${account_index} for: <input type="text" name="query"
value="*">
<input type="submit" width=200 value="Submit Query">
<input type="hidden" name="search_type" value="all">
<input type="hidden" name="template" value="results4.tmpl">
</form>
```

This search form, within an HTML template, looks like this in a browser:

Search the ${account_index} for: *
Submit Query

In this form, the following are VTL variables: `nav_search_uri`, `account`, `view`, `query`, `account_index`, `search_type`, and `template`. The form action values designate the URL to display. All the values for the variables are resolved when the CGI performs a query. Hence, the user does not actually see the word `account_name`, but rather an actual name for the given account. There are two hidden variables: `search_type`, and `template`. These constrain the query and are not visible to the end user in the search template. However, once the query is submitted, these values are visible in the browser URL. Once the user clicks `Submit Query`, the result template displays with this example URL in the browser:

```
http://qa/vss-bin/vss_SR.exe/my_acct/tutorial?query=*&search_type=all
&template=results4.tmpl
```

There are multiple places where you can set values for Virage Template Language variables. In the HTML form example, the `nav_search_uri` variable is set in a configuration file. The CGI looks up variable values in several places before committing a query to the search engine. For a description of the locations in which you can set variables and the parsing order, see "Where to set variables" on page 42 and "Variable parsing sequence" on page 46.

Note: For additional examples of how to compose queries with HTML hidden forms, see the online Virage Template Language tutorial. Create an account based on the `all_accounts tutorial` view in the Account Manager web pages to view the tutorial.

Search and results templates

There are two main types of variables: search variables for controlling queries and results variables for controlling what is displayed. The query control variables specify the search parameters the server performs. The results variables control how information retrieved from the server displays in a browser. Typically, as a web developer, you create two templates: a search template to control the query parameters and a results template to control and select the display. In the search template, you can set the query

control variables in an HTML form. The values can be determined by user input, a configuration file, the template, or in several other locations.

You can actually provide the code for the search and results template within the same file. Any template file can provide the HTML forms or links for queries as well as the formatting for the current results.

Figure 2–2 illustrates the how the search and results templates fit into the client-server architecture. When a user visits a web page in the application, the web server invokes the vss_SR.exe CGI. The CGI builds a hash table that contains all the variables and their values (from configuration files) and substitutes values defined the search template.

The user sees the rendering of a web page that should include a search box. The page that displays the search box is the search template. It is the HTML page that contains HTML forms and embedded Virage Template Language query variables.

When the user clicks to submit a search, the HTML form and query variables are sent to the web server, which invokes the CGI. The CGI, in turn, queries the underlying index where the metadata is stored. The CGI returns results (in a Perl hash table) and dynamically generates HTML pages using a results template to format and render results to a client browser. The results display in a static HTML page (with no embedded Virage Template Language syntax).

The CGI programs use XML to communicate directly with the solution server. As a template designer, you do not need to know XML to communicate directly to the core solution server. You only interact with the CGI programs to process queries. The CGI programs perform tasks, such as efficient querying or providing support for a media player.

In order to communicate with the CGI programs, you can use the Virage Template Language, designed for that purpose. The template language consists of variables, modifiers, and tokens for conditional statements. If you want to perform more complex customizing, you can extend functionality with Perl access functions as well. (Chapter 3 describes the Virage Template Language syntax.)

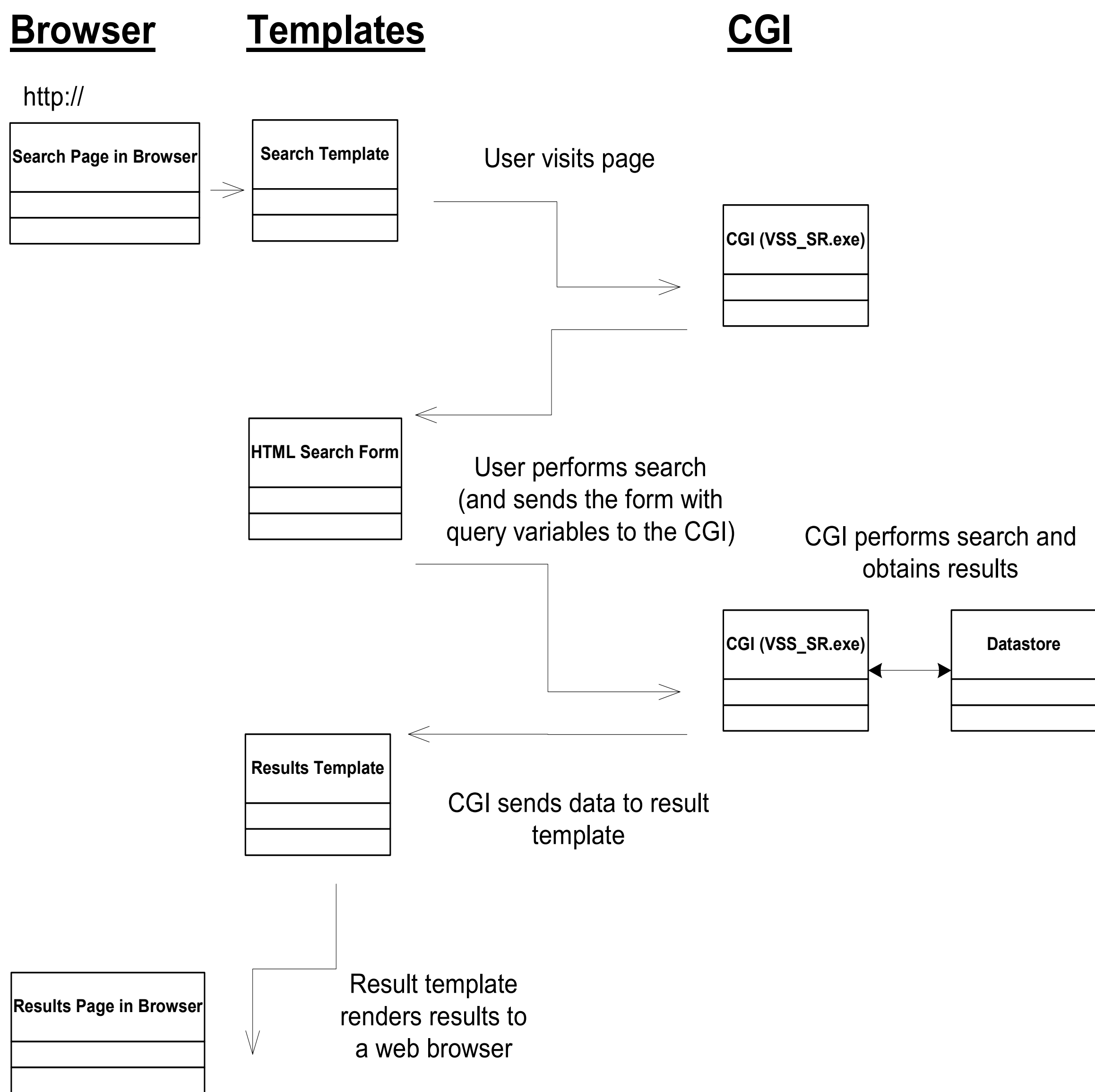


Figure 2–2: Template-CGI workflow within the solution server

What do search and results templates look like?

The following are descriptions of a search and result template.

Example: template with embedded variables and VTL statements

This example illustrates a sample HTML template with embedded Virage Template Language conditional statements and variables. The Virage Template Language variables and token are colored in blue text.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Variable substitution and VTL_IF example</title>
  </head>

  <body>
    <p>The value of the account_index variable is
${account_index}.</p>
<VTL_IF NAME="edit_enabled">
    <p>This paragraph becomes part of the returned data only if the
      edit_enabled variable exists and is nonzero.</p>
</VTL_IF>
<VTL_IF NAME="search_type" VALUE="videos">
    <p>This paragraph becomes part of the returned data only if the
      edit_enabled variable is equal to "videos".</p>
</VTL_IF>
  </body>
</html>
```

In this example, when a query is submitted, the CGI creates a result array and combines the data using the Virage Template Language conditional `VTL_IF` and variables statement to produce web pages. As the CGI combines data, it performs substitutions for variables, such as `${account_index}`.

This example includes Virage Template Language variables and two VTL_IF statements. The first VTL_IF statement allows the enclosed content into the CGI output, if the value supplied exists or it is not zero. The second VTL_IF statement allows the enclosed content into the CGI output, if the search_type variable is equal to videos.

The output of the CGI using this example template would be as follows (if account_index is equal to sample_index, edit_enabled is equal to 1 and search_type is not equal to videos.):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Variable substitution and VTL_IF example</title>
  </head>
  <body>
    <p>The value of the account_index variable is sample_index.</p>
    <p>This paragraph becomes part of the returned data only if the
      edit_enabled variable exists and is nonzero.</p>
  </body>
</html>
```

Example: results template that displays multiple results

This example template illustrates how to use the Virage Template Language to display multiple results on a page. To request multiple results (for example, for all videos whose transcript contains a particular phrase) you can use a VTL_LOOP statement to loop through an array of results.

The following sample template file uses a VTL_LOOP statement, but this sample illustrates an error.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>VTL_LOOP example with error</title>
  </head>
```

```
  <body>
    <p>The results come from video ID ${video_asset_id} and
      ${asset_id}.</p>
    <p>These are clip titles that contain query matches:</p>
    <ul>
<VTL_LOOP NAME="results">
      <li>${clabel_Title}</li>
      <li>${cinfo_duration:c}</li>
</VTL_LOOP>
    </ul>
  </body>
</html>
```

In this example, the CGI program tries to interpret the value for the variable `${video_asset_id}` but cannot find it at the top level of the result hash array. So the CGI replaces the variable with nothing. Then, the CGI loops through an array called `results` and returns values for each `${clabel_Title}` or clip label title within the array.

This template includes an error that illustrates the concept of scope. The template refers to the variable `${video_asset_id}` whose value exists only inside the results loop. The value of this variable cannot be determined outside the loop. These values are accessible to an inner loop. When the CGI program creates a result hash array, the CGI program adds the query parameters to the outer loop or array of the results. Variable data from outer loops is available in inner loops, but data contained in inner loops is not available to outer loops. Data from an outer loop, however, can be overridden in an inner loop to take on a different value.

The `cinfo_duration` variable is normally returned in milliseconds. However, with the `:c` modifier, the CGI converts the duration values to another clock format. In the case of the `:c` modifier, the output takes on the format that is defined by the `clock_format` query control variable. (The sample view configuration files set the `clock_format` variable to a specific value.) There are other modifiers you can use to instruct the CGI to covert data, such as to JavaScript readable code with the `:j` modifier. You can define the behavior of the `:c` modifier (in a configuration file). For more information on the Virage Template Language modifiers, see "Using variable modifiers" on page 73.

The data that the CGI renders to a web browser looks like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>                                      Empty results
    <title>VTL_LOOP example</title>
</head>
  <body>
    <p>The results come from video ID  and  .</p>
    <p>These are the titles of the clips that contained matches to
your query:</p>
    <ul>
      <li>E-Bonds</li>
      <li>00:02:13.45</li>
      <li>News Briefs</li>
      <li>00:02:11.21</li>
      <li>IPO Forecast</li>
      <li>00:02:09.01</li>
      <li>AOL-Time Warner</li>
      <li>00:02:21.55</li>
      <li>Apple's Comeback</li>
      <li>00:02:46.33</li>
    </ul>
  </body>
</html>
```

Variable substitution and example result hash

When a request is submitted, the CGI programs perform simple substitutions to derive values for the variables. The CGI programs pass data for a request to the solution server using a robust XML language. (The CGI programs return information to template designers in pre-defined Virage Template Language return variables.) For example, the variable `search_type` can constrain a search to the following values: `clips`, `videos`, `all`, `docs`, and `videos,docs`.

The variables must be defined someplace for the CGI programs to know which value to use to perform substitution. You can define variables in several locations, such as within the template or in configuration files. The various locations in which you can define variables and, more importantly, the

sequence in which they are parsed is discussed in "Requesting information to display" on page 36.

When a query is submitted to the solution server, the CGI programs create a result hash, combined with the templates, to produce web pages. The result hash contains values or arrays of values. An example result hash looks like this:

```
$VAR1 =
      {
          . . .
          'date_locale' => '',
          'edit_enabled' => 1,
          'results' =>[
       {
       'video_asset_id' => 1,
       'clabel_Title' => 'E-Bonds',
       },
       {
       'video_asset_id' => 1,
       'clabel_Title' => 'News Briefs',
       },
       {
       'video_asset_id' => 10,
       'clabel_Title' => 'IPO Forecast',
       },
       {
       'video_asset_id' => 10,
       'clabel_Title' => 'AOL-Time Warner',
       },
       {
       'video_asset_id' => 10,
       'clabel_Title' => 'Apple\'s Comeback',
       },
       ],
          'query' => 'the',
          'view' => 'edit',
          'account_index' => 'sample_index',
       . . .
        };
```

Each result hash can contain values or references to an array of inner hashes. Each CGI combines the data from the hash it creates with the referenced template file to create the CGI output that is sent to a browser to be rendered.

Requesting information to display

This section discusses what you need to know to request information to display. You request information by passing query control parameters, that define what it is you are seeking, to a CGI program. In order to request information in the templates, you need to know which Virage Template Language query control variables to use, where to specify them, and in what sequence a CGI program parses them from the various locations you can use.

Which query control parameters to use

The Virage Template Language includes many possible query control parameters (described in "Query control variables for VSS_SR" on page 87). However, to construct most queries, the basic parameters that you would use most often include:

- `template`

  Use to determine which template to use when displaying results

- `query`

  Use to specify the string for which to search in the index. (The parameter `fetch_records`*n* determines which records are actually retrieved from the database.)

- `asset_id`

  Use to restrict the results to the ID of the identified clip or video

- `video_asset_id`

  Use to restrict the results to the ID of a single video. Each clip has the same video ID as the video to which it belongs. You can use this constraint to retrieve only clips from a specified video. Not needed if `asset_id` is specified

- `fetch`*n_track name*

  If a tracks exist, use to determine the tracks to gather once a search is executed. A query searches for matching fields. A fetch gathers or collects the information, often a subset of the query. For example, you can query for a term within the closed-caption text and fetch corresponding images. Or, you can query for clip labels of a certain title and fetch

the corresponding closed-caption text to display. For an example using `fetch`, see the online Virage Template Language tutorial.

- `fetch_records`*n*

  Use this to specify to display either all records in the results or only those that match a query

- `max_results`

  Use to determine the maximum number of results to display

- `max_pages`

  Use to determine the maximum number of pages to return

- `search_type`

  Use to determine the type of asset on which to return

- `action`

  Use for some Perl CGI programs to refine the action performed. (This applies only to some Perl CGIs.)

The actual values you can use for each Virage Template Language query parameter are defined in "Search variables" on page 86. (For the sample search and edit views, the only actual required query parameter is the `template` parameter and it has a default value of `search.tmpl` that is defined in the view configuration files. If you do not specify to use another template, this one is used.)

Query control examples

The following examples illustrate how you can use some of the VTL query control variables to customize or build your own templates. Included are several small examples illustrating how to construct a URL with different query parameters, as well as a complete advanced query template example you can paste and view in a browser.

Note: In the following examples, the parameters to the CGI search program are shown starting with a slash. For example, the full URL

http://*server.domain.com*/vss-bin/vss_SR.exe/sample/edit?query=Virage

is shown as: `/sample/edit?query=Virage`

In addition, long lines of code are shown broken into multiple lines, but you should not include any space in the actual URL.

Basic syntax examples

Here are some basic Virage Template Language examples on how to construct URLs within templates.

Example 1: To display a default screen in the browser, insert the following in your Virage Solution Server URL or template:

```
/sample/edit
```

This defines the account to `sample` and the view to `edit`. The Virage Solution Server CGI uses a template file that is defined in the sample view configuration file variable `template` to render a page and display results.

Example 2: To display a query with results template, use this URL:

```
/sample/edit?query=Virage;template=results.tmpl
```

This defines the account to be `sample` and the view to `edit`. The results template file is now defined by the URL variable to `results.tmpl`. Defining the variable `query` queries on the word `Virage`. Results are displayed in the output template. Since the search is not restricted to asset type, all asset types are returned.

Example 3: To display a restricted query with a user variable, use this URL:

```
/sample/edit?query=Virage;search_mode=clips;search_type=docs;template
=results.tmpl
```

The account, view, query, and template variables are identical to Example 2. In addition, setting search_type to docs restricts the returned data to these assets only. The user variable search_mode controls template display and has no effect on the actual search.

Example 4: To fetch data from a single asset, use this URL:

```
/sample/edit?video_asset_id=1;search_mode=clips;search_type=docs;temp
late=results.tmpl
```

In this example, the query variable has been removed and replaced with a search for all assets with a video_asset_id of 1. Again, setting search_type to docs restricts the returned data to these assets only, that is, all assets with an ID of 1.

Note that since no highlight string is specified in this URL, no text records are returned. The query control variable is used as a default if highlight_string is not set. In this example, neither one is specified, so there is no true highlight string.

To return text records you must specify the corresponding fetch query control variable. The sample Edit view automatically sets the fetch variable to text_Closed Caption as a default in the view_config.xml file.

Example 5: To display text records that match a highlight string, use this URL:

```
/sample/edit?video_asset_id=1;search_mode=clips;search_type=docs;temp
late=results.tmpl;highlight_string=Virage
```

Setting highlight_string to Virage causes any text records with the word Virage to be displayed. Text records that do not have the highlight word are not returned.

Example 6: To display text records and highlighting matching words, use this URL:

```
/sample/edit?video_asset_id=1;search_mode=clips;search_type=docs;temp
late=results.tmpl;highlight_string=Virage;fetch_records=all
```

Setting fetch_records to all causes all records to be fetched, including ones without the highlight term.

Advanced query template example

This is a complete query example that you can copy and paste and view in a web browser. The example illustrates how to search for specific fields (all data, any video field, or any text track) and sort by asset ID and relevance score. In the example, users can choose the order in which to display the results.

To view the file in Virage Solution Server, create a file called `example.tmpl` and paste the sample code into that file. Place the `example.tmpl` file in a view directory, such as: `vs/conf/accounts/sample/edit/example.tmpl`. To view the file, view the following URL:

http://*localhost***/vss-bin/vss_SR.exe/sample/edit?template=example.tmpl**

There are embedded comments further explaining the example.

```
<html>
<head>
    <title>Example template</title>
</head>
<body>
<form name="searchform" action="${nav_search_uri}/${account}/${view}"
method="get">
<!-----------------------------------------------------------------
  Set the values of the following variables to the desired search
behavior
  ----------------------------------------------------------------->
<input type="hidden" name="template"    value="example.tmpl">
<input type="hidden" name="search_type"    value="videos">
<input type="hidden" name="query_array_op" value="and">
<!------------------------------------------------------------------
  Associate the three input fields with three specific fields in the
index.
  ------------------------------------------------------------------>
<input type="hidden" name="query_field1" value="VIR_VDF_ALL_DATA">
<input type="hidden" name="query_field2" value="VIR_ANY_VID_FIELD">
<input type="hidden" name="query_field3" value="VIR_ANY_TEXT_TRACK">
<!-----------------------------------------------------------
 Here are the text entry fields that allow the user to enter the
desired values for any of the fields.
  ----------------------------------------------------------------->
All Data:        <input type="text" name="query1"
                  value="${query1}"><br>
```

```
Any Video Field:  <input type="text" name="query2"
                  value="${query2}"><br>
Any Text Track:  <input type="text" name="query3"
                  value="${query3}"><br>
<p>
<!-------------------------------------------------------------
  Here is a simple drop-down that contains a couple options for
sorting the results. Results can be sorted by the asset ID or the
relevance score. For more information on sorting, see "Sorting by a
specified field" on page 54.
  -------------------------------------------------------------->
Sort By:
<select name="sort1">
<option value="VIR_ASSET_ID_FIELD"
        <VTL_IF NAME="sort1" VALUE="VIR_ASSET_ID_FIELD">
            selected
        </VTL_IF>>
Asset ID
<option value="VIR_RELEVANCE_SCORE"
        <VTL_IF NAME="sort1" VALUE="VIR_RELEVANCE_SCORE">
       selected
        </VTL_IF>>
       Relevance
</select>
<p>
<input type="submit" name="submit" value="Search">
<p>

</form>
<hr>
<!-------------------------------------------------------------
  This loop simply lists the assset IDs of the first "page" of
results. If the results are sorted by their relevance score, then the
score is also printed. The code checks if "score" is present in the
results array and determines whether results are sorted by relevance.
If "score" is present, the results are sorted by relevance.
  ------------------------------------------------------------->
<p>
<VTL_LOOP NAME="results">
    Asset ID: ${asset_id}
<VTL_IF NAME="score">(Relevance: ${score})
</VTL_IF><br>
</VTL_LOOP>
</body>
</html>
```

For additional advanced query examples, see the sample Advanced Search view.

Where to set variables

You can define values for query control variables in several locations. This allows you to create URLs visible to customers that are short, user-friendly, and do not contain long query control parameters. You can specify query control variables to send to the CGI programs in four places: URLs, HTML forms, configuration files (a server-side configuration file, `vss_config.xml`, an account configuration file, or a view configuration file), or within a `user.pm` Perl module that exists per each account.

If you create several templates that set the same variable to the same value, you can define the variable once in a configuration file for the view. Then you can reference the variable in the template files for that view. This is similar to setting and using a style in a Cascading Style Sheet. If you need to modify the value, you only need to modify the value once in order for the change to take effect throughout the templates. Each view has one configuration file, a `view_config.xml` file. If you want to define a variable for an entire account, which may consist of many views, you can define the variable in the account configuration file. The *Virage Solution Server Administrator's Guide* describes how to use the configuration files.

Since you can define variables in multiple locations, you must be aware of the order in which the CGI parses these locations so that you understand which values supersede all other places you might have set the same variable. The parsing sequence is discussed in "Variable parsing sequence" on page 46.

Specifying variables in the URL

You can specify any query control variable within the URL to define each query. Place query variables as name-value pairs after the question mark within the URL. Separate each name-value pair with an ampersand (&) sign or a semi-colon (;). For example, the URL

```
http://virage/vss-bin/vss_SR.exe/my_account/my_view?template=search.t
mpl;asset_id=2
```

specifies the executable CGI program as vss_SR.exe, the account as my_account, the view as my_view and a query that specifies a template of search.tmpl and an asset ID of 2. (The account and view locations are specified before the question mark to allow administrators to restrict access to these directory locations. Administrators cannot restrict access to anything specified after the question mark. To restrict access to the directory locations, you can use your web server security. For more information, see the *Virage Solution Server Administrator's Guide*.)

In another example URL

```
http://server/vs/cgi-bin/vss_SR.exe/myacct/myview?myvar=myvalue
```

the variable myvar is to set to myvalue.

Setting variables in the URL overrides any variables set in the template_vars sections of either the account or view configuration files but it does not always override variables defined in the account configuration file account section. The parsing order of variables is discussed in "Variable parsing sequence" on page 46.

Specifying variables in an HTML form

You can specify any query control variables within an HTML form. Place query variables as name-value pairs as follows:

```
<input type="hidden" name="template" value="results.tmpl">
```

the variable template is to set to results.tmpl. This overrides any variables set in the URL but it does not always override variables defined in the configuration files. The parsing order of variables is discussed in "Variable parsing sequence" on page 46.

Specifying variables in configuration files

Use the configuration files to define variables that you want to use repeatedly and that rarely change. For information on the configuration files, see the *Virage Solution Server Administrator's Guide*. Within the configuration files, define the variables as name-value pairs. Within the configuration files, there are four places you can define variables:

- In the section of the account configuration file called account

Any variables that you set in this location take effect for the entire account. They cannot be overridden in the view configuration file or in the URL.

- In the section of the account configuration file called `template_vars`

  Any variables that you set here are the default for all views within the account. However, they can be overridden in the view configuration file, in the URL, or in a Perl access function.

- In the section of the view configuration file called `template_vars`

  Any variables that you set here will take the default for the view. Variables defined here can be overridden in the URL or a Perl access function.

When you define a variable within a configuration file, you can use it within a template file with the same syntax as any variable: `${variable_name}`.

There is a parsing order for the various configuration file locations that you can use to define variables. The specific locations and their parsing order is included in the discussion on "Variable parsing sequence" on page 46.

Note: If you set a query variable in a configuration file, but do not set a query variable in a template file, the vss_SR.exe CGI does not query the core server and underlying search engine. This prevents the CGI from making unnecessary queries to the underlying server.

For example, if you set in the view_config.xml file the variables

```
<var name="query5"       value="MGM" />
<var name="query_field5" value="clabel_studio" />
<var name="query_op5"    value="eq" />
```

And do not set any query variable in the template, the CGI does not query the server. If any query is contained in the template, the template query, such as <var name="query1" value="Warner"/>, as well as the query contained in the configuration file is sent to the search engine. If query_array_op is set to "and", the search is restricted to results that contain both MGM and Warner. If the query_array_op is set to "or", the query searches for either MGM or Warner records.

In this manner, no queries are sent to the core server if the template does not contain a query.

Specifying variables in user.pm

For the template designer familiar with Perl, the Perl module user.pm is another location where you can define variables. The user.pm file is discussed in more detail in "User functions" on page 83. Here it is important to note that you can define variables within several functions in the user.pm file to extend functionality. You can use a pre_search() function to modify variables in a request for information and a post_search() function to modify information in the results hash before the CGI program combines the data with the template to produce web pages.

Possible uses for pre_search() could include a last minute re-direction to another template or to change the format of values that a user has entered. Possible uses for post_search() include adding a cookie or an ad banner based on specific results or using a log file to log query results. Or you can

use the `post_search()` function to process returned text, such as closed caption, to remove extra spaces. You can also use `post_search()` to direct local results to play from a high-resolution proxy and remote requests to play from a low-resolution proxy.

If you use the `user.pm` file to define variables, keep in mind the parsing order for the variables you use. Since the Perl function `pre_search()` is called before the search executes, you can modify any query control variables that have been set elsewhere, such as in a configuration file.

Variable parsing sequence

As you design templates and modify configuration files, keep in mind the following order in which the main solution server CGI search program uses and modifies variables. For example, if you declare a value in one location, such as in a configuration file while the same variable is defined elsewhere, you may get unexpected results, if you do not understand the parsing sequence.

Figure 2–3 illustrates the sequence in which the main solution server CGI program uses and modifies system variables. Figure 2–4 depicts this sequence with more detail on how the CGI processes information and produces a template.

1. Account and view values from the extra path information of the URL

   For more information, see "Account and view variables" on page 86.

2. URL query variables

   For more information, see "Query control variables for VSS_SR" on page 87.

3. Form variables from an HTTP POST command.

4. Default values from the `template_vars` section of the view configuration file

5. Default values from the `template_vars` section of the account configuration file

6. Default values from the `template_vars` section of the `vss_config.xml` configuration file

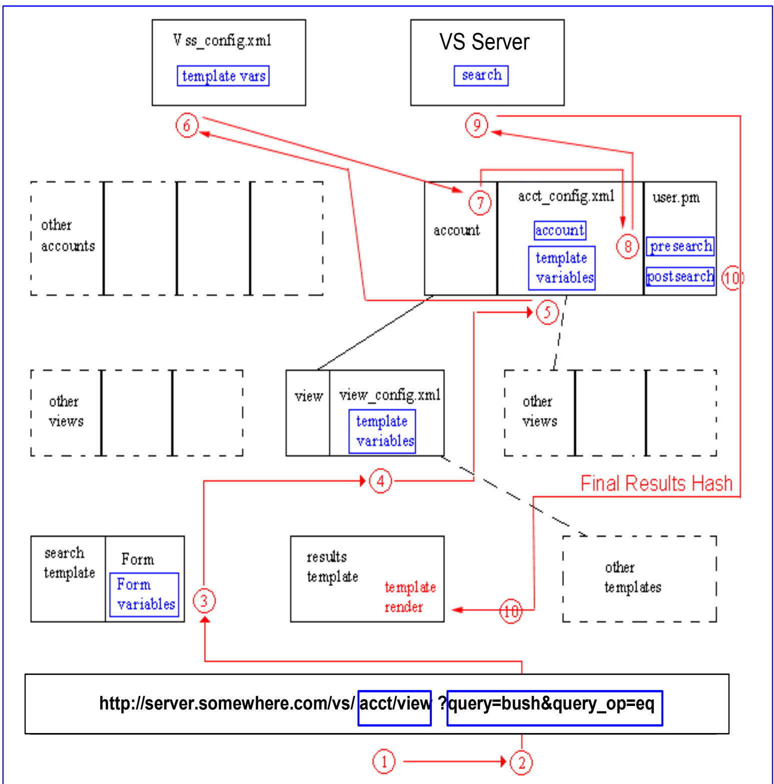


Figure 2–3: Variable parsing order

7. All values from the `account` section of the account configuration file

   If the same variable is defined more than once, variables are overridden with the value defined later in the parsing order. The definition that appears last is used, except for variables defined Step 7. Variables defined in the `account` section of the account configuration file always override the variables with the same name defined in Step 1 through Step 6.

8. The user Perl function `pre_search` ()

   For more information, see "Perl access functions" on page 83.

9. The search is executed. Once a search is executed based on current variable settings, search results are placed into a results hash table.

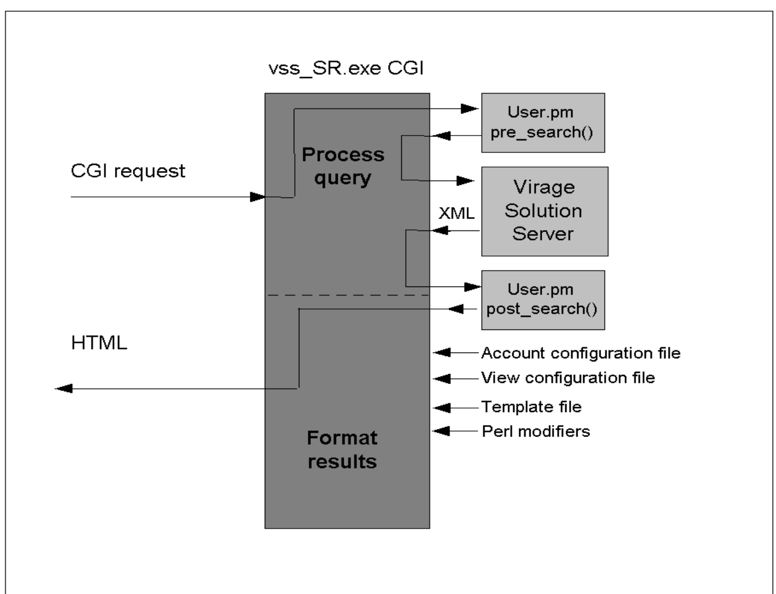


Figure 2–4: Process for invoking templates

10. The user Perl function `post_search()`

    For more information, see “Perl access functions” on page 83.

11. Variables are inserted into the template

At each step, the variable values that the `vss_SR` CGI obtains are added to the results hash. You can set many template variables merely by including them in the URL. Or, you can set defaults for certain variables, such as sort order, to have a wider scope, by setting variables in configuration files. Table 2–1 reviews the variable scope.

Table 2–1: Variable scope

Desired scope	Location
All accounts	`vss_config.xml` configuration file
Per account	`acct_config.xml` (all views within an account)

Table 2–1: Variable scope (*Continued*)

Desired scope	Location
Per view	`view_config.xml` (all templates within a view)
Per template	Hidden form variables within the template
Just the current web submission to the server	Directly on the URL, in the browser

Sample tasks for customizing templates

Following are some examples for performing certain tasks. For more examples, see the online Virage Template Language tutorial. Once you install the solution server SDK, create an account with a view that is based on the tutorial. Use the Administration web interface to create the view.

Creating new templates

Create a text file called `my_template.txt` and place it in the `conf\`<*account*>`\`<*view*>`\my_template.txt` directory for the account and view for which you want to create the template.

To view your new template, point your browser to the following location:

`http://`*host***`/vss-bin/vss_SR.exe/`***account***`/`***view***`?template=my_template.txt`**

For Solaris, use `vss_SR` (without the `.exe`).

Specifying the template to render

You specify the template with the `template` query control variable. You can set this variable with any of the methods described in “Where to set variables” on page 42:

- As part of the URL
- As an HTML form variable
- In a configuration file
- In `pre-search()` in the `user.pm` file

Modifying background colors

You can modify the background colors for the sample templates by changing a variable in the `view_config.xml` file for the view. In the sample `view_config.xml` file, the variable `color_view_body_background` controls the background color for the sample templates.

The sample templates ship with this variable set to the hex value `#EEEEEE` as the default background color. In the `view_config.xml` file, the following line indicates the color assignment:

```
<var name="color_view_body_background" value="#EEEEEE"/>
```

Once you set the color hex value in the view configuration file, you can use the `${color_view_body_background}` variable in each template file (within the `<body>` tag) to render the hex value for the desired color in the `bgcolor` attribute.

For example, if you change in the `view_config.xml` file the following lines to a different hex value such as:

```
<var name="color_view_body_background" value="#FFCC33"/>
```

The `${color_view_body_background}` variable renders light yellow-orange in all of the templates for that account.

The sample templates also use the ${color_<*description*>} variables to designate other colors, not just the background in the `<body>` tag. Therefore, if you change the assignment in the view configuration file, all instances of the ${color_<*description*>} variable are affected. If you want to use a different color for those other cases, you can create a new variable in the view configuration file and use it instead for those instances.

You can also modify the background colors for the Administration pages in the `vs\accounts\admin\vss.css` file. Within this file, modify the following syntax:

```
body
{
    color: black;
    background: #CDDDE9; /* light blue*/
}
```

Modifying the size of the embedded player

To change the size of the embedded player, you can change the width and height in the viewer template files. Depending on the size you choose, you may want to adjust sizes as well in the JavaScript file that pops up the viewer to adjust to large players. You can also adjust the proportions of the frame that contains the player in relation to the bottom frames (the navigation bar and content) in a frameset template.

To change the size of the embedded player, change the width and height attributes inside both the `<embed>` and `<object>` tags on the player templates. The `<embed>` tag affects the size in Netscape browsers and the `<object>` tag affects the size in Microsoft Internet Explorer browsers. The sizes, in pixels, in the `<embed>` tag should match those in the `<object>` tag.

The sample templates use the following sizes:

```
width="192"
height="144"
```

Modify the player templates. For example, for the sample Search view, modify the following player templates:

```
conf/accounts/sample/search/viewer_incl_mplayer.tmpl
conf/accounts/sample/search/viewer_incl_realplayer.tmpl
```

If you adjust the size of the player to a large size, you may need to increase the size of the page from which that player is viewed. Since the viewer launches a separate window from the results page, you can adjust the viewer page size by changing the width (and height) attributes in the JavaScript that opens the viewer window. Edit the viewer function in the `vss.js` file for the pertinent accounts. The `vss.js` files are located in the following location: `vssdocs/accounts/`*account_name*`/`*account_view*`/`. Adjust the width in the following syntax:

```
{
  var viewer = window.open(URI, 'viewer',
  'directories=no,height=718,location=no,menubar=no,resizable=yes,' +
  'scrollbars=auto,status=yes,toolbar=no,width=452,' +
  ...
}
```

If you adjust the height in the JavaScript file, it adjusts the height of the entire viewer, not just the frame that contains the player. Thus, you might prefer to adjust the height of the player frame instead. This method gives you more control over the proportional size of the player frame in contrast to the other frames in the viewer. To make these adjustments, modify the following frameset template:

conf/accounts/*account_name*/*view_name*/**viewer_frameset.tmpl**

In the following line, adjust the default 170 value to adjust the pixel height of the top frame:

```
<frameset
    rows="170, *"
```

For example, to adjust the height of the top frame to 420 pixels, set the frameset to `rows="420, *"`.

Displaying multiple proxy links on a results page

Each link on the results page can point to a single but separate player page (using a distinct template). You need to use a template page that plays the specified proxy type. Syntax for embedding each player differs by media player manufacturer and by browser manufacturer.

Alternatively, each link can launch a stand-alone media player. Again, the specifics of how to do this is different for each media player. Have the link point to a `.ram` file to launch the RealPlayer, `.mov` for QuickTime, `.asx` to launch Windows Media Player, etc.

Displaying additional text tracks

Because most text tracks contain a lot of information, the solution server does not return track data as part of the query results unless data from a

track is specifically requested. You can specify track data to return by setting a fetch query control variable using any of the methods described for setting query control variables. The sample templates have fetch1 set to text_Closed Caption in the view configuration files of the search and edit views. You can fetch data from all three types of tracks: text, image, and binary.

For example, to display a text track called text_OCR and another one called text_Speech, set the following variables: fetch3=text_OCR and fetch5=text_Speech. (The numbering fetch3 and fetch5 are not related to the order or tracks.) Use the text track names that are in the VDF file. You can specify multiple fetch tracks. For information on using the Virage Template Language search variable fetch, see "fetch1..n" on page 90.

You can display the information by looping through the text results as follows:

```
<VTL_LOOP NAME="text_OCR">
    ${timein_msec}
    <VTL_LOOP NAME="hitext">${plain}${highlight}</VTL_LOOP>
</VTL_LOOP>
<VTL_LOOP NAME="text_Speech">
    ${timein_msec}
    <VTL_LOOP NAME="hitext">${plain}${highlight}</VTL_LOOP>
</VTL_LOOP>
```

Inside the text track loop, the variable ${timein_msec} provides the start time for the track entry in milliseconds. (You might want to add :c modifier to convert milliseconds to a different time format. See the modifier ":c" on page 73.) The hitext loop provides part of the track entry: first text and then the term to highlight. (For example, you could emphasize the highlight term by wrapping it in tags.) You can customize the highlighting by editing the HTML tags or accompanying CSS.

Searching by dates and date ranges

You need to specify dates in one of the following formats: YYYY-MM-DD (2001-05-26) or MM-DD-YYYY (05-26-2001). If you need to specify a different date format, you can use the Perl function pre_search() to convert date values to one of the accepted formats prior to the query. For example, your search template can contain separate drop-down boxes for the month, day, and year. You can combine the values into a single string into one of the required formats.

You can use these operators on date values: `eq`, `gt`, `geq`, `lt`, `leq`, `ne`. You can use multiple constraints on the same fields.

For example, here is an example that illustrates how to search on date range that allows users to specify an optional minimum date and an optional maximum date for the Airdate field. This example assumes the customer is searching for assets based on the Airdate field.

```
<b>Airdate</b>:<br>
Starting: <input type="text" size="16" name="query1" value=""><br>
<input type="hidden" name="query_field1" value="vlabel_Airdate">
<input type="hidden" name="query_op1"    value="geq">

Ending:   <input type="text" size="16" name="query2" value=""><br>
<input type="hidden" name="query_field2" value="vlabel_Airdate">
<input type="hidden" name="query_op2"    value="leq">

<!-- To restrict a search to videos, use this the following code:
<input type="hidden" name="search_type"  value="videos">
-->
```

The operands are defined in “Query control variables for VSS_SR” on page 87.

Sorting by a specified field

You can specify to sort with the following control variables: sort, `sort1..sort`*n* (the sort fields to use, in order 1 through *n*), and `sort_dir` with `sort_dir1..sort_dir`*n*. Use ”`asc`” or “`desc`” to specify an ascending or descending sort.

Here are some guidelines for which fields you can use for sorting.

- You can sort by any video field, for example `vlabel_Title`
- You can sort by any clip field, for example, `clabel_Segment`
- You can sort by any special query fields, such as VIR_VDF_CREATION_DATE or VIR_VIDEO_ID_FIELD. You cannot sort by a special value that represents a group of values, such as VIR_ANY_VID_FIELD.
- You cannot sort results by text track values, such as closed-caption text or teletext.

For example, this example sorts video clip results by a specific value. The user can determine which of several fields determines the sort order and whether to sort by ascending or descending order.

```
Sort Clips By: <select name="sort1">
<option value="clabel_Title" selected>Clip Title
<option value="clabel_Dek">Clip Dek
<option value="clabel_Segment">Clip Segment
</select>

Sort Order: <select name="sort_dir1">
<option value="asc" selected>Ascending
<option value="desc">Descending
</select>
```

Modifying the proxy bitrate and format chooser

The sample templates allow the user to select a bitrate and format to stream media. Once the user selects preferences, they are stored in a cookie in the browser. The player plays at the bitrate closest to the speed the user selects. The player only plays the user-selected format if there is a proxy that exists in that format. If a proxy in another format exists, but the user did not select that format, the proxy does not play.

You can modify the choices presented to the user in the `view_config.xml` file for each view.

For example, you can set the following player variables to `"no"` to remove the choice from the bitrate and format chooser.

```
<!-- ** configurations for player bitrate/chooser ** -->
<var name="real_media_56k" value="yes" />
<var name="real_media_100k" value="yes" />
<var name="real_media_220k" value="yes" />
<var name="real_media_300k" value="yes" />
<var name="windows_media_56k" value="yes" />
<var name="windows_media_100k" value="yes" />
<var name="windows_media_220k" value="yes" />
<var name="windows_media_300k" value="yes" />
```

For more information, see the *Virage Solution Server Administrator's Guide*.

The player preference templates include the files playprefs.tmpl, redirect.tmpl, viewer_frameset.tmpl, viewer_incl_realplayer.tmpl, viewer_incl_mplayer.tmpl, viewer_mplayer.tmpl, viewer_realplayer.tmpl, and JavaScript functions in the vss.js library.

The redirect template redirects the player page of choice. There is a script included along with a JavaScript onLoad() function call to check for a browser cookie that contains the user preferences. The viewer templates call a Perl module called ProxySelect.pm that chooses the proxy. Each template file that contains an embedded player calls this module with the following syntax.

```
<vtl_script name="Virage::VTL::ProxySelect">
<vtl_call name="Virage::VTL::ProxySelect::set_user_proxy">
```

The ProxySelect.pm module contains the function set_user_proxy(). This function obtains the proxy to use from the user bitrate and format preferences stored in a cookie.

You can also specify the bitrate and format within the VTL_CALL. For example:

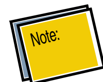
```
<VTL_SCRIPT name="Virage::VTL::ProxySelect">
<VTL_CALL name="Virage::VTL::ProxySelect::set_user_proxy"
bitrate="350" format="asf">
```

For more information on VTL_SCRIRT and VTL_CALL, see “Accessing a Perl module from a template file: VTL_SCRIPT” on page 80 and “Calling user functions within a template: VTL_CALL” on page 81.

Note: You can set your templates to play proxies from one of several proxy locations stored on other servers. To do so, edit the acct_config.xml file to point to the locations where the proxies are stored. For more information on setting the CDN setting, see the *Virage Solution Server Administrator’s Guide*.

Customizing playlist functionality

The playlist feature enables users to play multiple selected clips, videos, or audio files in a sequence. Users can play these assets dynamically or you can play assets that are stored in a database.



Note: The sample playlist plays only files that were originally uploaded to VSS as VDF files.

As a web developer, you can modify the sample templates, or create your own templates, and add an advertisement banner, for example, as a interstitial, to weave within the playlist sequence. For example, you can add a custom clip to a playlist at the beginning, middle, or end of a sequence or add interstitial logos in between each clip.

This section reviews the playlist implementation in the sample templates and illustrates how to add an interstitial file to a playlist.

The playlist Perl module and CGI

The sample templates call the following Perl module and CGI to invoke playlist functionality: `Playlist.pm` and `playlist.pl`.

The `Playlist.pm` module contains one function, `create_playlist()`, and handles the proxy information for each asset within a playlist, such as the time formats, playlist start and end time, and sorting of results by the sequence order the user chooses. The `playlist.pl` CGI program triggers the playing of the playlist and routes the MIME type, for example. The `playlist.pl` CGI outputs the playlist in the format (such as SMIL) determined in the `action`.

You can use one of the following action commands to support different formats. Use the `SMIL` format to support RealPlayer and `ASX` for Windows Media. Other formats are for earlier versions of browsers.

```
action=smil;template=smil.tmpl : playing smil file  (default)
action=asx;template=asx.tmpl: playing asx file
action=ram;template=ram.tmpl: playing ram file
```

```
action=rpm;template=rpm.tmpl: playing rpm file
action=rt;template=rt.tmpl: playing realtext file
```

Your `action` must be consistent with the `action` parameter for playlist to function. If you do not specify a MIME type or a template with the action, the template renders with the content_type of `application/smil` (by default).

Invoking the playlist in the templates

There are several implementations of the playlist feature in the sample templates. Depending on which parameters you pass to the `playlist.pl` CGI, the CGI invokes the playlist in one of several scenarios.

- From within the Search and Edit views, you can add clips or videos to a playlist and click Play Playlist to play the sequence of selected clips or videos. The playlist launches dynamically from within the Edit and Search view templates
- From within the Folders view, you can play existing Folder contents that are saved in the Folders database. The playlist is launched for a current folder stored in a database.
- From within the Folders view, you can preview a playlist—play selected clips or videos prior to saving them to a Folders database. The playlist is invoked to preview a playlist prior to saving the information to the database.

Playing a playlist from the Edit and Search views

In the first scenario, the playlist plays dynamically from the Edit and Search templates. As the playlist information is not saved to a database, the playlist information is gathered on the fly by a JavaScript function into the `playlist_ids` parameter. Each asset ID is separated by a colon. In the sample `viewer_incl_realplayer.tmpl` viewer template, the syntax for playing a playlist looks like this for the Internet Explorer browser:

```
<param name="src" <VTL_IF name="playlist_ids">
value="${action_playlist_uri}/${account:u}/${view:u}?action=smil;play
list_ids=${playlist_ids};template=smil.tmpl;asset_type=${asset_type}"
```

The `playlist_ids` parameter is an array list of asset IDs. The `playlist_ids` parameter exists if the assets are not stored in a database.

The JavaScript function stores the IDs dynamically on the client as the user adds assets to a playlist. The playlist information is not persistent in this case.

Instead of calling the `playlist_ids` parameter, you can also list the IDs with this syntax: `playlist_ids=3:2`. You can use this syntax for an embedded player or a stand-alone player.

The playlist calls customized files for each format. For example, there are sample `smil.tmpl` and a sample `asx.tmpl` files. As a template developer, you can choose the streaming format to play. You can create your own streaming template file.

Playing a playlist from the Folders database

The base URL for playing from a folder that is saved in a database looks like this:

```
${servername}/vss-bin/playlist.pl/${account}/${view}
```

You can use this URL either in the `src` tag for embedded player or launch the stand-alone player. To launch a stand-alone player, you can pass the URL as a CGI call. The sample template files illustrate this within an embedded player. For example, from the sample `viewer_incl_realplayer.tmpl` file (or other format viewer include template files) the playlist CGI is called to play a playlist, the URL to the CGI is passed as a `src` value, and the CGI generates a `SMIL` file to send to the player.

Within the sample `viewer_incl_realplayer.tmpl` file, the playlist CGI is called with this syntax for the Internet Explorer browser.

```
<param name="src" <VTL_IF name="playlist_ids">
value="${action_playlist_uri}/${account:u}/${view:u}?action=smil;play
list_ids=${playlist_ids};template=smil.tmpl;asset_type=${asset_type}"-
<VTL_ELSE_IF NAME="bin_id">
value="${action_playlist_uri}/${account:u}/${view:u}?action=smil;temp
late=smil.tmpl;bin_id=${bin_id};play_bin=1;max_results=100"
```

Within the `src` value, there is a conditional statement. The `VTL_ELSE_IF` statement illustrates this scenario. If a `playlist_ids` parameters exists, the first URL is used (which is the first scenario). If there is no `playlist_ids` parameter, the statement checks for a `bin_id` parameter. The `bid_id` refers

to a unique Folder within the Folders database. If the `bin_id` exists, the URL within the `VTL_ELSE_IF` statement is used to play the playlist.

The following parameters are required for the URL for playing a saved folder:

- `bin_id=?` that specifies which folder to play
- `play_bin=1` that specifies to `user.pm` to bypass the authentication for playing folder. Within the URL, you must specify `play_bin=1` to set this value to `TRUE` to play the URL.
- `action=?` that specifies which streaming format to use. The most commonly used values include: SMIL and RAM for RealPlayer and ASX for Windows Media.
- `template=?` that specifies which streaming file to use. The most commonly used value include: `smil.tmpl`, `asx.tmpl` or `ram.tmpl`. This value must be consistent with action value. If `action=smil`, then `template=smil.tmpl`.

For the Netscape browser, the syntax, within the same viewer template file, looks like this:

```
<VTL_ELSE_IF NAME="bin_id">
src="${action_playlist_uri}/${account:u}/${view:u}?action=smil;templa
te=smil.tmpl;bin_id=${bin_id};play_bin=1;max_results=100"
```

In this example code, the `action_playlist_uri`, `account`, and `view` variables values are derived from the configuration files. The `:u` encoding ensures that any special characters are encoded for proper URL format.

Note: The Folders view uses the Edit viewer templates when a Folders view is created. To edit the templates in the master templates, edit the viewer files in the Edit view. Once a view is created, you can edit the viewer files in the Folders view to modify the viewer for a specific Folders view.

Previewing a playlist from the Folders view

This scenario plays, or previews, a playlist prior to saving the playlist information in the Folders database. In other words, the user plays the playlist prior to clicking Save to Database to store the information in the Folders database. Since there is a lot of information to pass to the URL, the sample templates use a POST command. This method launches a stand-alone player. You cannot use this method to play within an embedded player since the embedded player `src` value expects a URL. The sample templates use this syntax to invoke the stand-alone player to preview a playlist prior to saving it to a database.

The client CGI launches the player from this command:

```
<form name="pre_bin" action="/vss-bin/playlist.pl/${account}/${view}"
method="POST">
<input type="hidden" name="prebin_contents" value="">
<input type="hidden" name="template" value="">
<input type="hidden" name="action" value="">
<input type="hidden" name="play_bin" value="1">
<input type="hidden" name="play_prebin" value="yes">
```

The following input values are required for preview folder:

- `play_prebin=yes` indicates this is to preview a folder. The `play_prebin` value must be set to `yes` to preview the folder playlist.
- `prebin_contents=?` that contains all the client side data of the folder. This is a list of asset IDs.
- `action=?` that specifies which streaming format to use. The most commonly used values include: SMIL for RealPlayer and ASX for Windows Media or RAM.
- `template=?` that specifies which streaming file to use. The most commonly used value include: `smil.tmpl`, `asx.tmpl` or `ram.tmpl`. This value must be consistent with action value. If `action=smil`, then `template=smil.tmpl`.

In the sample templates, the values for the `prebin_contents`, `template`, and `action` variables are derived from the `cb_custom.js()` function. You can view an example in the file `cb_edit.tmpl`.

Adding an interstitial to a playlist

To add an interstitial file, such as a logo GIF file, to a sequence, you can modify the format template file. For example, you can add an interstitial to the sample `smil.tmpl` file.

```
<vtl_script name="Virage::VTL::ProxySelect">
<vtl_call name="Virage::VTL::ProxySelect::set_user_proxy"
format="ram">
<vtl_script name="Virage::VTL::Playlist">
<vtl_call name="Virage::VTL::Playlist::create_playlist">
<smil>
<head>
</head>
<body>
<!-- you can add an interstitial here as a header to the playlist-->
<vtl_loop name="results">
<!-- you can add an interstitial here ->
<img src="images/logo.gif" dur="5.5s" />
<!-- You must specify a dur value to specify the duration ->
<VTL_IF name="asset_type" value="videos">
<video src="${proxy_url}" clip-begin="0ms"
clip-end="${vinfo_duration}ms" />
<VTL_ELSE_IF name="asset_type" value="clips">
<video src="${proxy_url}" clip-begin="${cinfo_timein_msec}ms"
clip-end="${cinfo_timeout_msec}ms" />
</VTL_IF>
</vtl_loop>
<!-- you can add an interstitial here as a footer to the playlist-->
</body>
</smil>
```

Within the image `src`, the `dur` value is required to specify the duration of the interstitial. The first `VTL_CALL` chooses the `RAM` format. The second `VTL_CALL` obtains the values for the variables required for the body of this `SMIL` file. The `VTL_IF` statement determines whether the assets is a video or clip and plays either the video or clip.

You can enter an interstitial file before, after, or inside the `VTL_LOOP`. However, do not specify the interstitial within the `video` tag. If you enter the interstitial before the `VTL_LOOP`, the logo appears once, followed by video or clip. If you enter the interstitial after the loop, the video or clips play first and then the logo appears. If you enter the interstitial within the loop, the logo appears multiple times. You can add as many interstitials as needed.

The result `SMIL` file of this CGI looks like this:

```
<smil>
  <head>
  </head>
  <body>
   <img src="images/logo.gif" dur="5.5s" />
     <video src="file:///C:/Program
Files/virage/vs/vssdocs/sample_media/nws116_56k.rm"
           clip-begin="105848ms"
           clip-end="477948ms" />
    <video src="file:///C:/Program
Files/virage/vs/vssdocs/sample_media/nws116_56k.rm"
           clip-begin="486315ms"
           clip-end="613548ms" />
 </body>
</smil>
```

Developing templates with folders

You can write templates that render the folders you created into a viewable, paginated display. All of the data stored in the solution server index for each of the folder assets is available to your template.

You can see an example folder template after you create a folder in the user-interface. Click Folders on the navigation bar to go to the Folder Manager, select a folder, and click View. The template that is rendered is the sample Folders view and is called `cb_view_share.tmpl`.

The following URL parameters are essential to rendering a folder template:

- `bin_id=?`

  Specifies the unique numeric folder ID, as displayed in the folder manager.

- `max_results=?`

  Specifies how many assets to display on a page. If not specified, the value is 10.

- `bin_offset=?`

  Specifies where in the folder to begin the display. (This is how pagination is accomplished.) If not specified, the value is 0.

The base URL for rendering a folder into a template looks like this:

${nav_search_uri}/${account}/${view}?template=*template*&bin_id=*bin_id*

When the solution server renders a folder template, it inserts the following template variables and arrays, along with the result variables returned by vss_SR.exe:

- `${bin_asset_count}`

  Total count of assets in the folder

- `${bin_name}`

  Name of folder

- `${bin_description}`

  Description of folder

- `${bin_id}`

  Unique numeric ID for folder

- `${bin_owner}`

  Owner (creator) of the folder

- `${bin_modification_date}`

  The date the folder was last modified

- `${bin_creation_date}`

  The date the folder was originally created

- `${bin_annotations}`

  `VTL_LOOP` array of the following variables that are annotations for the folder: `${field_name}` and `${field_value}`

For each asset in the results `VTL_LOOP` array, these are the following template variables:

- `${bin_asset_id}`

  Unique numeric ID of this folder asset

- `${vss_asset_id}`

  Numeric ID of the solution server index asset

- ${override_timein_smpte}

  Replacement start time for a clip, in SMPTE format.

- ${override_timein_msec}

  Replacement start time for a clip, in milliseconds.

- ${override_timeout_smpte}

  Replacement end time for a clip, in SMPTE format.

- ${override_timeout_msec}

  Replacement end time for a clip, in milliseconds.

- ${bin_order}

  Position of this asset within the folder.

- ${bin_asset_annotations}

  VTL_LOOP array of these variables for annotations for the asset: ${field_name} and ${field_value}

Folders and the user.pm module

There are special considerations regarding the account user.pm module and the Folders view.

In order to support the rendering of folders into templates, the user.pm module includes code in the pre_search() and post_search() functions. This code calls two main libraries of the solution server, the user authentication library and the folder database library. The hooks in user.pm are used to authenticate users and fetch folder data from the folder database for rendering.

Authentication is performed for any URL that includes a "bin_id=" parameter or any URL that includes "id_check=1" parameter. The authentication library retrieves the browser cookie and sets values for the user and account. The hooks to the folder database API, Virage::ClipBins::ClipBinDB, retrieve folder data from the database to paginate, sort, and add folder variables to the result hash.

For your application to use Folders, these hooks must be present in the user.pm module.

Note: For anonymous users (users who are not logged in) to render a Folders view template, edit the `user.pm` module for the pertinent account and delete the call to user authentication where "bin_id" is a parameter. This does not affect any other authenticated folder management operations. This only impacts rendering a Folders view templates to view or play (any URL that includes `"/vss-bin/vss_SR.exe/`*actount*/*view*`"`).

Using the JavaScript library

Virage provides several JavaScript functions in the sample templates. JavaScript is useful for constructing forms and windows. The sample templates use JavaScript to define radio buttons and to launch the viewer windows, among other tasks. There are also several predefined JavaScript methods that ship with the solution server installation that facilitate customizing the templates.

The JavaScript functions for the sample templates are defined for each view in the following subdirectories. These are the master copies used to create views.

vs/exten/vssext/sample/master_templates/sample_acct/views/
view/**vssdocs/**
vs/vssdocs/accounts/admin
vs/vssdocs/accounts/sample/*view*

You can call a JavaScript function as part of an HTML tag such as in the radio buttons in the sample `search.tmpl` template. In this case, the script, which is defined earlier in the template, sets a number of HTML form values as a result of user selection. This is the sample template code:

```
<input type="radio" name="search_type" value="videos"
onclick="Video()"> Videos
```

Many JavaScript functions are defined in `vss.js`. Other JavaScript functions are defined within the templates themselves. Many of the sample templates also use a JavaScript function called `URI_Builder` to format URLs. For information, see Appendix A “The URI Builder Script.”

Here is an example JavaScript function, `set_hidden_search_fields()`, that is set in `vss.js` and is called by the `results.tmpl` and `results_header.tmpl` template files.

```
function set_hidden_search_fields()
{
  var which = document.searchform.chooser_menu.selectedIndex;
  var chooser_menu = document.searchform.chooser_menu[which].value;
  if (chooser_menu == "AllFiles")
  {
    document.searchform.search_type.value = 'videos,docs';
    document.searchform.query1.value = '';
  }
  else if (chooser_menu == "AllClips")
  {
    document.searchform.search_type.value = 'clips';
    document.searchform.query1.value = '';
  }
  else if (chooser_menu == "AllFilesClips")
  {
    document.searchform.search_type.value = 'all';
    document.searchform.query1.value = '';
  }
  else if (chooser_menu == "VideoFiles")
  {
    document.searchform.search_type.value = 'videos,docs';
    document.searchform.query1.value = 'VIR_CAT_VIDEO';
  }
  else if (chooser_menu == "VideoClips")
  {
    document.searchform.search_type.value = 'clips';
    document.searchform.query1.value = 'VIR_CAT_VIDEO';
  }
  else if (chooser_menu == "AudioFiles")
  {
    document.searchform.search_type.value = 'videos,docs';
    document.searchform.query1.value = 'VIR_CAT_AUDIO';
  }
  else if (chooser_menu == "AudioClips")
  {
    document.searchform.search_type.value = 'clips';
    document.searchform.query1.value = 'VIR_CAT_AUDIO';
  }
  else if (chooser_menu == "WordProcessorFiles")
  {
```

```
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_DOC';
  }
  else if (chooser_menu == "Presentations")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_PRESENTATION';
  }
  else if (chooser_menu == "Spreadsheets")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_SPREADSHEET';
  }
  else if (chooser_menu == "PDFs")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_PDF';
  }
  else if (chooser_menu == "HTMLFiles")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_HTML';
  }
  else if (chooser_menu == "TextFiles")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_TEXT';
  }
  else if (chooser_menu == "Images")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_IMAGE';
  }
  else if (chooser_menu == "Other")
  {
    document.searchform.search_type.value = 'docs';
    document.searchform.query1.value = 'VIR_CAT_OTHER';
  }
  else
  {
    document.searchform.search_type.value = 'all';
    document.searchform.query1.value = '';
  }
}
```

The `set_hidden_search_fields()` function sets the appropriate hidden query and search type values for the media type that the users choose in the search page.

The file `incl_categories.tmpl` maps the query values, such as `VIR_CAT_IMAGE`, to the user-friendly display for the category terms, such as `Image`.

To add new data types to the drop-down menu on the sample template, add the search type and query values to this function, as well as to the `incl_categories.tmpl` file. Modify the search form in the search template file.

Working with international templates

If you are developing templates for an international audience, there are several things you should note.

Set character encoding to native codepage

In the view configuration files, set `char_encoding` to the native codepage for the locale you want to support. For example, to display Korean characters, use the native codepage `ks_c_5601-1987` (for Windows) or `EUC-KR` (for Solaris).

You can also set `char_encoding` to UTF-8. However, Netscape 4.x browsers do not properly display form input field characters in UTF-8. If you want to support Netscape browsers, use your native codepage. In the view configuration file, you can also set the appropriate `template_lang` variable for the language you support. Netscape 6.x supports UTF-8.

If you want to use Administration pages (that are localized) in Netscape 4.x browsers, you should also set the Admin view configuration file `char_encoding` variable to your native codepage. If you use UTF-8, Netscape 4.x may not handle input, such as for a description of an account, that is not in the native codepage for your browser. You can also use Microsoft Internet Explorer or Netscape 6.x browsers.

Avoid font-family specifications

As you develop localized templates for a non Latin-1 character set, Virage recommends that you do not specify styles in the font-family in your style sheets. For example, avoid font styles, such as serif and sans serif. The Virage Solution Server Western sample templates use font styles. To customize these for a non-western character set, remove the font-family in the style sheets as well as in the templates. For example, remove `face="Verdana, Arial, sans-serif"` from the `font` tags.

However, if you know which fonts work well with your character set, you can use those fonts.

Compatibility with pre-release 2.2 templates

If a template that was created prior to release 2.2 uses multibyte data for VDF label field names, the clip and video editing features within Virage Solution Server do not work properly with Netscape 4.x. For example, Netscape 4.x may not display certain Edit Select and Multi Select fields properly. The Netscape 4.x JavaScript engine does not properly handle multibyte characters strings. To address this problem, `user.pm` maps the original VDF label field names to a single byte string prior to rendering the `editor_vlabels.tmpl` and `editor_clabels.tmpl` templates.

To allow for backwards compatibility and use older templates created prior to release 2.2, the `editor_vlabels.tmpl` and `editor_clabels.tmpl` template files use a hidden field called `template_version`. If the `vss_Edit.pl` script finds this hidden field in a template, the `vss_Edit.pl` functions (edit_vlabels and edit_clabels) map the single byte placeholder back to the original VDF label field name values prior to sending an edit command to Virage Solution Server.

3 Using the Virage Template Language

The Virage Template Language (VTL) is a powerful language that allows you to customize the functionality and appearance of the templates without programming. The VTL is incorporated into several of the Virage Solution Server CGI programs and the Administration user interface. Any templates that these CGI programs use can incorporate syntax and functionality that the VTL provides.

The VTL syntax includes tokens, variables, variable modifiers, and Perl access functions. You use tokens, variables, and modifiers within the templates and you can access Perl functions from within a template, modify existing Perl functions called each time a template is rendered, or create your own functions.

VTL includes syntax for displaying variables from a search result, control statements for nesting templates, looping of output, and conditional display (in addition to other functionality). In addition, special variables can control the parameters of a search command and the type of data that is returned and displayed.

This chapter reviews the Virage Template Language, including the syntax for using variables, tags, Perl access functions that you can use to modify variables, and system-wide variables.

Chapter 2 “Customizing the Templates,” describes how to use VTL to create new templates as well as how to customize the sample templates that are shipped as part of the Virage Solution Server. The tutorial that ships with the Virage Solution Server SDK provides additional examples.

Variable syntax

VTL substitutes values from variables. You can use variables in the template files with the following syntax:

${*VAR*}
${*VAR*:*MODIFIER*}

Variable names are case sensitive. For more information on using variables, see "Variable substitution and example result hash" on page 34.

Using variable modifiers

Table 3–1 describes the variable modifiers that are built into the vss_SR.exe CGI program. If a CGI program finds a modifier in the modifier table, the CGI attempts to call the user Perl function of that name. (For more information on the Perl functions, see "Perl access functions" on page 83.) If no modifier is specified, the :h modifier is used for HTML encoding.

Table 3–1: Variable modifiers

Modifier	Description
:c	Use for clock (human-readable) encoding. For example: xxh xxm xxs (only applies to millisecond values). This is encoded using the printf style format string in the variable clock_format for localization. The first two format elements are for hours and minutes and must be an integer specifier as the %d specifier. The third element is for seconds and may either be an integer specifier as the %d specifier or a floating point specifier as the %f specifier.
:d	For date (Human-readable) encoding (only applies to date values). By default, dates are presented in ISO 8601 format (YYYY-MM-DD) but the :d format presents them in MM-DD-YYYY format. This is encoded using the strftime format string in the variable date_format for localization which allows customizing to other formats. For example: Aug. 24, 2001.
:h	Use for HTML encoding

Table 3–1: Variable modifiers (*Continued*)

Modifier	Description
:j	Use for JavaScript encoding. For example, use to control back slashes so that they are properly interpreted in JavaScript.
: m	Used to expand macro substitution. For example, see "Macro substitution" on page 75.
:u	Use for URL encoding. Use to create a URL to properly encode various characters, such as a blank space.
:s	Use for Real Networks style SMPTE encoding (only applies to millisecond values)
:t	Use for text encoding. This is essentially a null operation and is used to override the default `:h` HTML encoding.
:q	For QuickTime encoding (hh:mm:ss:frames) format. For example, 02:25:30:24.5 means 2 hours 25 minutes 30 seconds and 24.5 frames (only applies to millisecond values).

You can use modifiers in sequence. For example, to convert from a time to clock format for inclusion in a JavaScript program, you could use the following code:

```
"${clabel_Date:d:j}"
```

Using VTL_ tokens

There are special tokens you can use to control looping, to include a file, to use conditional statements, to log template output, or to use a Perl script. All template tokens must appear within your template on a single line. That is, a line break is not allowed inside a token. Token names are not case sensitive. Tokens have similar notation as HTML tags.

<Token> .... *</Token>*

where Token is the name of the VTL token. Table 3–2 lists the VTL tokens. All VTL tokens include an attribute called `name` for which you assign a value.

The `name` attribute can take a literal string or a variable upon which the statement can perform an action.

Table 3–2: VTL tokens

VTL Token	Page
VTL_LOOP	page 77
VTL_INCLUDE	page 78
VTL_IF	page 79
VTL_ElSE	page 80
VTL_ELSE_IF	page 79
VTL_SCRIPT	page 80
VTL_CALL	page 81
VTL_LOG	page 82

For example, the VTL_LOOP token accepts an attribute for the name of an array in which to loop through. For example, to loop through the array called `results`:

```
<VTL_LOOP NAME="results">
```

The `VTL_INCLUDE` statement can accept a attribute name for a file to include in the template. For example:

```
<VTL_INCLUDE NAME="header.tmpl">
```

The `VTL_SCRIPT` statement takes a name of a Perl module to use. For example:

```
<VTL_SCRIPT name=”WordWrap.pm”>
```

And the `VTL_LOG` statement acts upon a name of a log file. For example:

```
<VTL_LOG name="logfile">
```

Macro substitution

You can use macros as part of a token attribute. For example, in the sample `viewer_mplayer.tmpl` file, you see the following example:

```
setCookie('${default_bitrate}asf','${account}');
```

The `default_bitrate` is defined in the `acct_config.xml` configuration file. (The `${account}` value is defined by `vss_SR`. It is not a defined in a configuration file.)

In another example, this code returns True if the value of `proxy_url` is the same as the value of `default_proxy`.

```
<VTL_IF NAME="proxy_url" VALUE="${default_proxy}">
```

As part of a string value, you can use the `:m` modifier to expand variable names inside a variable value. For example, if the variable `var` has the value xxx and `var2` has the value `${var}/yyy`, the template string `${var2:m}` would render as `xxx/yyy`.

For example, in the sample `view_config.xml` configuration file you see the following assignment:

```
<var name="label_v1" value="${vlabel_Program Name}"/>
```

The value of `label_v1` is derived from the schema and user selection when a user creates an account.

In the `results_result.tmpl` file, this variable is used with a modifier as follows:

```
<VTL_IF NAME="result_type" VALUE="clips">
  <b>${label_c1:m}</b><br>
<VTL_ELSE_IF NAME="some_empty_string" VALUE="${label_v1:m}">
  <b>${asset_Title}</b><br>
<VTL_ELSE>
  <b>${label_v1:m}</b><br>
</VTL_IF>
```

In this example, the following statement:

```
<VTL_ELSE_IF NAME="some_empty_string" VALUE="${label_v1:m}">
```

is equivalent to this statement:

```
<VTL_ELSE_IF NAME="some_empty_string" VALUE="${vlabel_Program
Name}">
```

The `some_empty_string` is not defined, so it is equal to an empty string. The `VTL_ELSE_IF` statement will return True if there is no value for `${label_v1:m}`.

Looping variables with VTL_LOOP

You can use the looping token to re-display a section of a template with several different variable values. Following is the syntax for the looping token:

```
<VTL_LOOP NAME="LOOPVAR"> ... </VTL_LOOP>
```

where *LOOPVAR* is the name of an array that contains values for the variable you are looping. Within the token loop, you must include the variables for the loop you want to display.

Once a query result is received from the search server, several arrays that contain the variable values are created. When the results are displayed, the text between the `<VTL_LOOP NAME="LOOPVAR"> ... </VTL_LOOP>` token is displayed repeatedly for each set of values in the *LOOPVAR* array.

For example, after a query for clip data, the data for each result is held in an array called `results`. In each of the results the variable `clabel_Description` holds the value of the clip label description for that result. To display the clip label descriptions, you would use the following example in your template:

```
<VTL_LOOP NAME="results">
      Clip Description: ${clabel_Description} <BR>
</VTL_LOOP>
```

This example would produce the following sample HTML output:

```
Clip Description: Boy was rescued out of icy river <BR>
Clip Description: Grandmothers allowed to see boy and lobby
 Congress to allow boy to return home to Cuba <BR>
Clip Description: President Clinton proposes to use $1.9 tril-
lion surplus to pay down public debt <BR>
```

Use nested `VTL_LOOP` statements to access data stored at deeper levels of the results data structure. For example, within each result is an array that contains each text record for the closed-caption track. This array is accessed by placing a `VTL_LOOP` for the text track inside the `VTL_LOOP` for the result. For example:

```
<VTL_LOOP NAME="results">
      Clip Description: ${clabel_Description} <BR>
    <VTL_LOOP NAME="text_Closed Caption">
      Clip text: ${text} <BR>
    </VTL_LOOP>
</VTL_LOOP>
```

See “Perl access functions” on page 83 for a description of Perl representation and access to template loop variables.

Including template files: VTL_INCLUDE

You can insert the contents of another template file or a URL into a template. Use the following syntax:

```
<VTL_INCLUDE NAME="filename.tmpl">
             error_template="error.tmpl"
             parser="yes|no|user_sub">
```

where `filename.tmpl` is a filename path relative to the directory of the current template. The `error_template` indicates a template to include if the other template is not available. The `parser` parameter indicates whether the main solution server CGI should parse the included template content or use a Perl subroutine on the content.

You can use `VTL_INCLUDE`, for example, for headers or footers, where the information is repeated on each page.

For example, create a `header.tmpl` file, and within each template file, include the statement: `<VTL_INCLUDE NAME="header.tmpl">`

You can also include a URL with the following syntax:

```
    <VTL_INCLUDE name="http://www.xxx.com/yyy"
                 timeout="seconds"
                 error_template="error.tmpl"
                 parser="yes|no|user_sub">
```

The template files include output from the URL. If a host cannot be contacted or the time out value is exceeded, the error template is includes instead.

By default, the output from the URL is not parsed by `vss_SR.exe`. In other words, the URL content is not parsed to interpret any VTL syntax or to interpolate any variables. To explicitly prevent the included file from being parsed, you can specify “`no`” for the parser value. You can also specify a user Perl routine accept the output from the included file or URL. The `user_sub` Perl subroutine is called with the same conventions for the `VTL_CALL` described on page 81. The named Perl subroutine is invoked with

a hash and variable stack parameters. The text of the included file or URL is stored in the parameter hash value.

Using conditional statements: VTL_IF, VTL_ELSE, and VTL_ELSE_IF

You can design your template in such a way to include a block of template mark-up language, depending on the value of a particular parameter.

VTL_IF

You can use the following Virage Template Language syntax:

`<VTL_IF NAME="`*CONTROL_PARAMETER_NAME*`">... </VTL_IF>`

The *CONTROL_PARAMETER_NAME* is your control parameter. If the parameter is given a value that is true for Perl, such as `1` or `foo`, then the block is included in the output. If the parameter is not defined, or given a value that is false for Perl, such as `0` or an empty string, the block is skipped.

For example, you can use the following syntax to display some text only if there are results for a particular query:

```
<VTL_IF NAME="n_results">
      You have some results
</VTL_IF>
```

The line “You have results” is only displayed if the value of variable `n_results` is non-zero.

You can also compare a control parameter to a specific value, in addition to 0 and 1, to determine behavior. To compare the control parameter, use the following syntax:

`<VTL_IF NAME="`*CONTROL_PARAMETER_NAME*`" VALUE="`*TEST_VALUE*`">... </VTL_IF>`

This test is true if the current value of *CONTROL_PARAMETER_NAME* is the same as the test value *TEST_VALUE*.

VTL_ELSE_IF

For simpler evaluation of multiple conditions, you can `VTL_ELSE_IF`.

For example:

```
<VTL_IF name="x">
     ... template for condition x ...
    <VTL_ELSE>
      <VTL_IF name="y">
       ... template for condition not x and y ...
      </VTL_IF>
    </VTL_IF>
```

can be written in this format:

```
<VTL_IF name="x">
 ... template for condition x ...
<VTL_ELSE_IF name="y">
 ... template for condition not x and y ...
</VTL_IF>
```

VTL_ELSE

You can include an alternate block of text in a VTL_IF statement with the <VTL_ELSE> syntax. For example:

```
   <VTL_IF NAME="n_results">
You have ${n_results} results
   <VTL_ELSE>
     You didn't get anything.
   </VTL_IF>
```

Note that you end the block with a </VTL_IF> tag, not with a </VTL_ELSE> tag.

You can use multiple VTL_ELSE_IF statements in one VTL_IF clause. This is easier to test multiple conditions without having to count the number of closing tags.

Accessing a Perl module from a template file: VTL_SCRIPT

Syntax:

<VTL_SCRIPT name=*module name*”>

This token enables you to have access to a Perl module from within a template file. You can use VTL_SCRIPT to load a Perl Module into a template. This token is similar to the #include statement in C programming.

You can use VTL_SCRIPT to load Perl modules from an account or a global library. For security, you can only load modules that are located in the account directory lib/VSS_Accounts/ or the lib/Virage/ directory.

You can also refer to packages located in these directories. For example:

```
<VTL_SCRIPT name="Virage::VTL::test::get"/>"
```

Calling user functions within a template: VTL_CALL

Syntax:

VTL_CALL name=“*subname*” *attr1*=”*attr1val*” ... *attrn*=”*attrnval*”>

You can use VTL_CALL to call any user function and associated hash table values from within a template file. Attribute names are restricted to letters, numbers, and underscore, and are converted to lower case.

The CGI will call the named function and pass two parameters—a reference to a hash table and a variable stack. The variable stack is an array reference of all of the hash values at the current template location. In this array, the first element consists of the global variables, the second element consists of the set of variables for this iteration of the first VTL_LOOP statement and all further elements are hashes of variables for each successive nested VTL_LOOP. The hash table contains a name-value pair for each of the attributes included in the VTL_CALL statement.

For example, if you specify “use Data::Dumper” in user.pm, you are requesting a call to the subroutine called Dumper to output to standard-out. You can also specify the following in a template file:

```
<vtl_call name=“Dumper” myarg1=”foo” ARG2=“BAR”>
```

to produce the following output:

```
$VAR1 = {
          'myarg1' => 'foo',
          'name' => 'Dumper',
          'arg2' => 'BAR'
        };
$VAR2 = [
          {
```

```
            'keyframe_url_base' =>
'/vs/binary_data/sample_index',
            'account_index' => 'sample_index',
            'char_encoding' => 'ISO-8859-1',
            'date_format' => '%Y-%m-%d',
            'fetch1' => 'text_Closed Caption',
            'template_lang' => 'en-US',
        # ... Other global variables ... #
          }
        ];
```

The $VAR2 contains the variable stack and $VAR1 contains the hash table with values passed as parameters to attributes of the VTL_CALL statement. Notice that the parameter ARG2 was changed to a lower case arg2, although the data values were not changed. All the returned values will display in the template.

You can use this, for example, if you have a function that reformatted closed-caption text to fix word wrapping, you could call the function from within a template and reformat the closed-caption text before rendering to a browser. You can alter any template results with any user function with VTL_CALL.

See also the discussion on "User functions" on page 83.

Troubleshooting and tracking to a log file: VTL_LOG

Syntax:

```
<VTL_LOG name="logfile"> ...template data to log ... </VTL_LOG>
```

The VTL_LOG call can help you troubleshoot or track data, such as usage data, to a log file. You can enclose template data within VTL_LOG tags and the data will be written to a log file in the vs/logs/ directory each time the VTL_LOG statement is called. The log file is written with a module within the solution server and will be locked to avoid problems with concurrent access.

You could use the VTL_LOG to track page access to send to a log file data each time a template is accessed. You can use this data for billing custom-

ers or to track usage for other purposes to help gauge frequency in which users are accessing a page.

Perl access functions

All template variables are stored in Perl hash tables. You can use Perl functions to modify these variables or to create new ones. To do so, you can extend a Perl module called `user.pm` that exists for each account. The subroutines in the `user.pm` module are called every time a page is rendered. There is a `user.pm` file in each subdirectory for each account in the `VSS_Accounts` directory. For example, for the sample account, the `user.pm` file is located in `VS/lib/VSS_Accounts/sample/user.pm`.

User functions

The sample `user.pm` file declares some predefined functions in the `VSS_Accounts::account::user` package that you can use in your templates. Table 3–3 describes these Perl access functions.

Table 3–3: User functions

Name	Parameter	Description
pre_search	Reference to variable hash	Use to modify search control variables before the search is executed
post_search	Reference to variable hash	Use to modify search result variables after the search is executed but before display
modifier	Variable value	Use to modify a string value, such as for a simple substitution. For example, `${var:mysub}` calls `mysub` with the current value of `var`

When `user.pm` calls the user functions, they refer to a hash table. In this hash table, keys correspond to variable names, and values are either scalars for simple variables or array references if the name corresponds to a loop

array. Within the array are a series of hash references, with each hash containing the variable values for that instance of the array.

To illustrate this, a subset of the data that is returned after a query is shown below, as output from Data::Dumper. The `$var1` represents the complete hash. The `'results'` is the results array.

```
$VAR1= {
          'account_display_name' => 'Sample',
          'results' => [
                 {
                   'video_asset_id' => 1,
                 'vlabel_Description' => 'Headline News',
                           'text_Closed Caption' => [
                                {
                            'text' => '>>>New York yesterday.',
                                  'timein_msec' => 328,
                                  'timeout_msec' => 20625,
                                  'timein' => '00:00:00:10',
                                  'timeout' => '00:00:20:19'
                                  },
                                  {
                            'text' => 'Mississippi blues sing',
                                    'timein_msec' => 20625,
                                    'timeout_msec' => 33500,
                                    'timein' => '00:00:20:19',
                                    'timeout' => '00:00:33:15'
                                   }
                            ],
                    }
           ],
};
```

hash
array
first (and only) array element, which is a hash
array
array entries, each a hash

In this code, the top level variable `account_display_name` has the value `Sample`. The loop `results` contains an array with one element (one result). In the hash table for that result, the video asset ID is `1`, the video description label is `Headline News`, and there is another array reference that provides variable values for the two text records in the result.

Perl function example

The solution server sample templates contain examples of how you can use the predefined user functions. This section illustrates an example using the `post_search` function to modify the proxy setting for a video. Note that you

can also modify the proxy settings in other manners, such as in the solution server sample edit pages or within the `user.pm` file with a simple substitution.

To modify the proxy settings in the `user.pm` file using the `post_search` subroutine, add the following `for` loop anywhere after the `my $result_hash = shift;` line:

```
sub post_search()
{
    my $result_hash = shift;
    #
    # Change the default proxy_url
    #
    for my $res ( @{$result_hash->{'results'}} )
    {
        if ( $res->{'proxy_url'} =~ /nws116lo\.rm$/ )
        {
            $res->{'proxy_url'} =
             'PROXY_URL/RM_FILE';
        }
        elsif ( $res->{'proxy_url'} =~ /tv421lo\.rm$/ )
        {
            $res->{'proxy_url'} =
                'PROXY_URL/RM_FILE';
        }
    }
}
```

where the *PROXY_URL* points to your proxy locations for your content and *RM_FILE* is the `.rm` file that you want to play. For example, `rtsp://realserver.virage.com/nws116lo.rm` and `rtsp://realserver.virage.com/tv421lo.rm.`

System variables

The solution server provides system variables that you can use in your template to refine a search and certain system variables that you can use to return results. This section reviews the search and result variables.

Search variables

The solution server provides several types of system variables you can use to refine a search. You can use variables that control the parameters of the query, such as the maximum number of results to return, the order in which the results are sorted, or the offset from which each result should begin to play. Variables can apply to the entire account, or they can be view specific. It is important to know the sequence in which variables are modified by the CGI program that renders search results. For more information, see "Variable parsing sequence" on page 46.

Account and view variables

Table 3–4 lists the account and view specific variables. For more information on the account or view configuration files, see *Virage Solution Server Administrator's Guide*.

Table 3–4: Account and view specific variables

Variable	Source	Description
account_display_name	Account configuration file (acct_config.xml)	Display name of account
account_index	Account configuration file (acct_config.xml)	Index used by this account to connect to the server
char_encoding	View configuration file template_vars section	Character encoding used for display
template_lang	View configuration file template_vars section	Language code sent on the HTTP line
content_type	View configuration file template_vars section	HTTP content type MIME code. Default is text/HTML

Query control variables for VSS_SR

To define a query in a search template, use the query control variables described in Table 3–5. For examples that illustrate how to use many of these control variables, see "Query control examples" on page 37.

Table 3–5: Query control variables

Name	Description
offset	Offset from the first set of results. Use offset to display the next set of results. For example, assuming `max_results` is set to 10 to return 10 results per page, you can set offset to 10 to obtain results number 11-20.
query, query1 ... query*n*	String values for which to search. The variable `query` is combined with the variables `query_op` and `query_field` to form a constraint. Additional constraints are created by matching `query1` with the variables `query_op1` and `query_field1`, and so on. The operand specified by `query_array_op` is placed between each constraint. Note that the default values of `contains` and `VIR_ALL_VDF_DATA` for `query_op` and `query_field` mean that setting the variable `query` to a text value will cause a search for that string in all text data.

Table 3–5: Query control variables (*Continued*)

Name	Description
query_op, query_op1 ... query_op*n*	Operand to use for searching with a similarly numbered query variable. Use operands to construct queries on appropriate data types. May be one of the following (operand description is in parenthesis): `eq` (equal) `eq_ignore_caps` (equal, ignore capitalization) `ne` (not equal) `gt` (greater than) `lt` (less than) `geq` (greater than or equal) `leq` (less than or equal) `contains` (contains any of the terms) `must_contain` (all results must include all terms) `phrase` (Finds a contiguous phrase) `typo` (Finds results with spelling variations) `soundex` (Finds results that sound like the term. English only.) `syn` (Finds synonyms) Defaults to `contains`. For examples, see “Using query operands” on page 93.
query_field, query_field1 ... query_field*n*	Field or special value to be searched. Defaults to `VIR_ALL_VDF_DATA`. For possible other special values you can use, see Table 3–6, “Special values,” on page 91.
query_array_op	Operand to be applied between each query condition. This may be `and` or `or` and defaults to `and`. For examples, see “Using query operands” on page 93.
highlight_string	Words in this string are highlighted in any returned results. Defaults to the words in the first query variable
account	Derived from the URL /vss-bin/vss_SR.exe/*account*/view. Defaults to `sample`
view	Derived from the URL /vss-bin/vss_SR.exe/account/*view*. Defaults to `edit`
asset_id	Restricts the search to the specified asset ID. Every video and clip has a unique asset ID.

Table 3–5: Query control variables (*Continued*)

Name	Description
video_asset_id	Restricts the search to the specified video asset ID. Each clip has the same video ID as the video to which it belongs. You can use this constraint to retrieve all clips from a specified video. Not used if `asset_id` is also specified
max_results	Maximum number of results to return per page, including both video and clip results.
max_pages	Maximum number of pages in a pagebar to compute
search_type	Constrains the type of search to `clips`, `videos`, `all`, `docs`, or `videos,docs` (There is no space between the word `videos`, the comma, and the word `docs`). The `videos,docs` option searches on every data type except for clips. For example, use `videos,docs` in combination with an advanced query for VIR_CAT_VIDEO, to obtain results that contain all video files. Use `videos,docs` in combination with an advanced query for VIR_CAT_AUDIO, to obtain results that contain all audio files.
max_highlights	Maximum number of highlighted records returned per hit
max_msec	For keyframe tracks, the maximum value of the first keyframe returned in milliseconds
min_msec	For keyframe tracks, the minimum value of the first keyframe returned in milliseconds

Table 3–5: Query control variables (*Continued*)

Name	Description
fetch1..*n*	Name of any text, image, or binary track to return once a search is executed. (written as `text_`*trackname* or `image_`*trackname*). By default, fetch1 is defined as text_Closed Caption. A query searches for matching fields. A fetch gathers or collects the information, often a subset of the query. For example, you can query for a term within the closed-caption text and fetch corresponding images. Or, you can query for clip labels of a certain title and fetch the corresponding closed caption to display. For an example using fetch, see the online Virage Template Language tutorial.
sort1..*n*	Names of any fields to be sorted. If the first character of the field is a `+`, the sort is ascending; if it is `-` it is descending. Sorting direction may also be controlled by the variable `sort_dir`. The default is ascending.
sort_dir1..*n*	Direction of sorting field `sort1`...`sort`*n*. This variable is ignored if sort direction is already controlled by the first character of the `sort` variable. This variable is useful if sort direction is controlled by a check box. If the `sort-_dir` variable is `-` or `desc` then sorting is in descending order, otherwise sorting is ascending.
fetch_records	Specify `hits` to return only those records in which highlighting occurs or `all` to return all records. This variable is replaced with `fetch_records`*n*. For compatibly with existing templates, you can use this variable or the `fetch_records`*n* variable.
fetch_records*n*	Specify `hits` to return only those records in which highlighting occurs or `all` to return all records. This sets the value for corresponding `fetch`*n* track. You can set `fetch_records1` to `hits` for one track and `fetch_records2` to `all` for another track. If there is no value set for `fetch_records`*n*, the value of `fetch_records` is used. If `fetch_records` is not set, the default of `hits` is used.
template	Name of the template to display.

Table 3–5: Query control variables (*Continued*)

Name	Description
xml_debug	If set, XML command and reply data will be stored in `xml_command` and `xml_reply`
xml_get_schema	If set, XML schema data is stored as a string in the variable `xml_schema` and as variables in the arrays cschema and vschema
asset_Category	The data type category of the asset
asset_MIME-Type	The MIME type associated with the asset category
asset_Title	The title for the asset
asset_Subject	The subject of the asset description
asset_Author	The author for the asset
asset_Keywords	Any keywords associated with the asset

The query control variable `query_field`, which defines the query, can be further refined with special values. For example, use the special value VIR_VDF_CREATION_DATE to search for only assets created on a specific date. You can also use most of the special variables to refine sorting order. Table 3–6 lists the special values.

Table 3–6: Special values

Term	Description
VIR_LABELSET_NAME_FIELD	Use to search the name of the label set that was used for logging and encoding
VIR_ASSET_ID_FIELD	Use to find or sort by a specific video or clip ID. Each video and clip is assigned a unique ID.

Table 3–6: Special values (*Continued*)

Term	Description
VIR_VIDEO_ID_FIELD	Use to find or sort by a video ID and all of its clips. For videos, the video ID is the same as the asset ID. For clips, the video ID is the asset ID of the video to which the clip belongs.
VIR_VDF_CREATION_DATE	Use to find or sort by videos created on, before, or after a certain date. Express in the format YYYY-MM-DD.
VIR_INDEX_DATE	Use to find or sort by videos that were indexed on, before, or after a certain date. Express in the format YYYY-MM-DD. Same as VIR_LOG_DATE.
VIR_LOG_DATE	Use to find or sort by videos that were indexed on, before, or after a certain date. Express in the format YYYY-MM-DD. Same as VIR_INDEX_DATE.
VIR_CLIP_START_MSEC_FIELD	Use to find or sort by clips whose start time (in milliseconds) meets a specific constraint
VIR_CLIP_END_MSEC_FIELD	Use to find or sort by clips whose end time (in milliseconds) meets a specific constraint
VIR_ANY_VID_FIELD	Use to search by all fields in the video label
VIR_ANY_CLIP_FIELD	Use to search by all fields in the clip label
VIR_ANY_TEXT_TRACK	Use to search by all text tracks
VIR_ALL_VDF_DATA	Use to search all of the data in an original VDF. This includes the video label, the clip labels, and all of the text track data. This also includes metadata stored in the asset_Title, asset_Author, asset_Keywords, asset_Subject, asset_MIME-Type variables.

Table 3–6: Special values (*Continued*)

Term	Description
VIR_ALL_DATA	Use to search all of the data includes in VIR_ALL_VDF_DATA. However, VIR_ALL_DATA also includes other information such as the original filename, asset ID, label set names, etc.
VIR_VDF_FILENAME	Use to search against the original filename of the VDF file. The name of the file is determined by the location of the file when it is added to the solution server. This includes the entire path of the file.
VIR_RELEVANCE_SCORE	Use only to refine sorting order. You can use this as a primary key only. You cannot query on this special value. Use to find a numeric value between 0 and 100 that indicates the relevance of the video or clip to the specified query. A higher value indicates a better match with the query.
VIR_CATEGORY_FIELD	Use to refine searching or sorting by the category of asset type. Possible values are listed and mapped to MIME types in the vss_ftype.xml file. These include: VIR_CAT_VIDEO, VIR_CAT_DOC, VIR_CAT_TEXT, VIR_CAT_PDF, VIR_CAT_SPREADHSEET, VIR_CAT_PRESENTATION, VIR_CAT_AUDIO, VIR_CAT_IMAGE, VIR_CAT_HTML, VIR_CAT_OTHER. See “Using the asset_ variables” on page 96.

Using query operands

Use query operands to construct complex queries. The `query_op` operand combines each query and the `query_array_op` operand combines multiple expressions. Combine query operands logically with the data type on which to perform the comparison.

Table 3–7 lists the Virage Template Language operands and the data types with which you can use to construct complex queries. For example, the `eq`, `gt`, `geq`, `lt`, `leq`, `and ne` operands are boolean query terms for numeric operations, such as for date comparisons. The `eq` operand works on numbers and the `gt` operator does not operate on strings, only on integers or dates.

The • indicates that the operand applies to the data type. The • ** indicates that the data type is treated as a string and is therefore applicable to the operand. The `eq` operator comparison is case-sensitive. For Select, EditSelect, and MultiSelect expressions, the value determines whether the operand is applicable. For example, if the Select field is a string, you can use the operands that apply to strings.

Table 3–7: VTL operands and pertinent data types

Data type/ Operand	String	Date	Integer	Float
contains	•	• **	• **	• **
must contain	•	• **	• **	• **
EQ	•	•	•	•
NE	•	•	•	•
GT	•	•	•	•
LT	•	•	•	•
GEQ	•	•	•	•
LEQ	•	•	•	•

Table 3–7: VTL operands and pertinent data types (*Continued*)

typo	•	N/A	N/A	N/A
syn	•	N/A	N/A	N/A
phrase	•	N/A	N/A	N/A
soundex	•	N/A	N/A	N/A

This is an example of a logical complex query, a combination of multiple query expressions:

query_field	query_op	query
clabel_Description	contains	"George Bush"

query_array_op= "and"

query_field1	query_op1	query1
clabel_Closed Caption	contains	"NYC"

query_array_op= "and"

query_field2	query_op2	query2
clabel_Date	qeq	"2002-04-01"

Using the contains and must_contain operands

You can use `contains` to define a search that seeks any one keyword and `must_contain` to search only for all of your terms. For example, if you use `contains` and search for "2002-03-22" the search engine returns all dates that contain the year 2002.

Here is another example. In this query, the CGI searches for all records in the closed-caption text track if *any one* of the keywords in the query (inky, blinky, pinky, or clyde) appear in the record.

query_field	query_op	query
text_Closed Caption	contains	"inky blinky pinky clyde"

Whereas, if you use `must_contain`, the query searches only for records in the closed caption text track that contain all keywords in the query field.

query_field	query_op	query
text_Closed Caption	must_contain	"inky blinky pinky clyde"

Therefore inky, blinky, pinky, and clyde all must be present in a closed-caption text record to produce a match.

Searching date fields

Specify dates in one of the following formats:

```
YYYY-MM-DD   (example: 1988-05-23)
MM-DD-YYYY   (example: 05-23-1988)
```

If you require a different format for your data, you can use the Perl function `pre_search()` to convert the date value to one of the accepted values. For example, you may choose to have separate drop-down lists for month, day, and year, you can combine the values into a single string and convert the format. You can also use JavaScript to combine the values into a single string and convert the format at the time the user submits the query to the CGI.

You can use the following operands on date values:

`eq, gt, geq, lt, leq, ne`

You can use multiple constraints on the same field, for example, to construct a date range query.

Using the asset_ variables

Here is an example of how to construct a query with the `VIR_CAT_CATEGORY` values. This example is from the `results_header.tmpl` template file. In

this example, the hidden HTML form variables restrict the search to any `asset_Category` that has the value of `VIR_CAT_PDF`.

```
<form action="${nav_search_uri}/${account}/${view}">
<input type="text" name="query" value="*">
<input type="hidden" name="query1" value="VIR_CAT_PDF">
<input type="hidden" name="query_op1" value="phrase">
<input type="hidden" name="query_field1" value=
"asset_Category">
<input type="hidden" name="template" value="results.tmpl">
<input type="submit" value="Search">
</form>
```

You can use the `asset_` variables as query control variables as well as to control result display. You can also add a media type that is not included in the sample templates. For more information, see "Asset type result variables" on page 103.

Result variables

Once a query is submitted, the CGI search program, `vss_SR.exe`, retrieves information from the database and renders these results for HTML display in a web browser. There are several types of information that can describe each result—variables that describe the entire set of results returned from a query and variables specific to each result.

The CGI program stores data for multiple results within arrays. The main array that contains all information is called the `results` array. As the CGI renders results, the program loops through each array to display the applicable value for a result.

For example, the CGI can loop through the `results` array and for each result display a different value for the result_type variable (whether the result is a clip or a video). To display this example in a web page, you can use the `VTL_LOOP` tag in a template file to loop through any array. For example, the following `VTL_LOOP` statement loops through the `results` array to display the correct type of result (clips, videos, or docs) per search result.

```
<VTL_LOOP NAME="results">
    <tr>
        <td> ${asset_id}</td>
        <td>
        <VTL_IF NAME="result_type" VALUE="clips">
            Clip (from video ${video_asset_id})<br>
```

```
        <VTL_ELSE_IF name=result_type" VALUE="videos">
            Video
        </VTL_ELSE>
           Document
        </td>
       <td><img src="${keyframe_url_base}/${keyframe_url}"></td>
    </tr>
</VTL_LOOP>
```

In the example, for each result, the asset ID and type are displayed in the browser using the `asset_id` and `result_type` variables within the `results` array. The keyframes are displayed using the `keyframe_url` variable, which must be appended to the variable `keyframe_url_base`, so that the web server can find the images.

The following tables describe the variables and arrays you can use in templates to customize the information to display for each search result. Table 3–8 describes variables that contain information on the entire set of results that can be returned from a query. Table 3–9 describes return variables that contain information specific to each result.

Table 3–8: Result variables for the entire set results

Name	Description
total_results	Total number of results found by this query
n_results	Number of results on this page
result_start	Number of the first result on this page. This will be equal to 1 + offset. For example, if the offset is zero `result_start` will be `1`.
result_end	Number of the last result on this page. This will be equal to `offset+n_results`.
next_page_query	If defined, the query parameters to go to the next page of results. Not defined if there are no further pages.
previous_page_query	If defined, the query parameters to go to the previous page of results. Not defined if there are no previous pages.

Table 3–8: Result variables for the entire set results (*Continued*)

Name	Description
xml_command	XML query sent to the solution server for debugging. Only set if `xml_debug` is set.
xml_reply	XML reply returned from the solution server for debugging. Only set if `xml_debug` is set.

For the variables that contain information specific to each result, you can use the `VTL_LOOP` statement to loop through the `results` array to access and display the values for each search result.

Table 3–9: Result variables specific to each result

Name	Description
asset_id	Asset ID of the clip or video. For videos, this will be the same as the video asset ID variable `video_asset_id`.
catalog_date	Date this asset was inserted into the index
keyframe_msec	In-time in milliseconds of the preview image keyframe for this result. If the user-defined preview images can not be set for this video, this variable will not be set.
keyframe_url	URL of the preview image for this result, relative to the binary data directory for the index
n_hit_lines	Number of the last hit_line in the text
result_type	Type of this result (`clips`, `videos`, or `docs`)
score	Relevance score of this asset. The score only applies if the primary sort key is VIR_RELEVANCE_SCORE.
vdf_filename	Original filename of the asset
video_asset_id	Asset ID of the video containing the result

Video and clip label result variables

Each result contains video information and, if this is a clip result, the clip information. This information is dependent on the label set for the index. Clip labels are prefixed by `clabel_`, and video labels are prefixed by `vlabel_`. The field name and appended to the clip or video prefix. For example, a label for a video title field is `vlabel_Title`, a label for a clip preview image field is `clabel_PreviewImage`, or a label for a clip description field is called `clabel_Description`.

Other asset types also contain `vlabel_` and `clabel_` label fields once the asset is uploaded to VSS and contained with a VDF file.

Proxy result variables

The CGI program stores proxy information for a default proxy in the `results` array. For proxy information specific to each asset, the CGI program stores proxy information in the `proxies` array. The proxy variables, such as `proxy_bitrate`, are the same whether the information is in the `results` array for the default proxy or in the `proxies` array. Table 3–10 describes the proxy variables.

Table 3–10: Proxy result variables

Name	Description
proxy_bitrate	Bitrate, in kilobits-per-second, at which the proxy streams
proxy_framerate	Proxy frame rate, per second, for the source video
proxy_height	Proxy height in pixels
proxy_mime_type	Proxy MIME type (such as `video/x-ms-asf` and `application/x-pn-realmedia`)

Table 3–10: Proxy result variables (*Continued*)

Name	Description
proxy_offset	Offset from which the proxy plays relative to the source video. The proxy offset value is the number of milliseconds between the start of the proxy encoding and the start of video metadata capture. It may be positive or negative. A positive offset means that the proxy video starts later than the start of the metadata capture.
proxy_url	URL to the proxy video location
proxy_width	Proxy width in pixels
proxy_is_default	`1` if this is the default proxy, otherwise undefined

Video information result variables

Video information about the VDF file is contained in result variables preceded by `vinfo_`. Table 3–11 describes the video information result variables.

Table 3–11: Video information result variables

Name	Description
vinfo_bitrate	Video bitrate
vinfo_character_encoding	Video character encoding
vinfo_creation_date	Video creation date
vinfo_duration	Video duration. This value is computed and is read-only.
vinfo_filepath	Video filepath
vinfo_fps	Video frames per second

Table 3–11: Video information result variables (*Continued*)

Name	Description
vinfo_height	Video height (in pixels)
vinfo_label_set	Video label set
vinfo_modification_date	Video modification date
vinfo_name	Video name
vinfo_standard	Video standard
vinfo_version_major	Video version major
vinfo_version_minor	Video version minor
vinfo_width	Video width field (in pixels)

Clip information result variables

If the query result is a clip, the clip time information is contained in variables preceded by `cinfo_`. Table 3–12 describes the clip information variables.

Table 3–12: Clip information result variables

Name	Description
timein	In-time in SMPTE
timeout	Out-time in SMPTE
timein_msec	In-time in msec
timeout_msec	Out-time in msec
duration	clip duration in milliseconds

Asset type result variables

The `asset_` metadata fields are used to describe multimedia documents. As multimedia assets are uploaded in an index, this information is entered into VDF files that are created. Table 3–13 describes these asset variables that are used to display the multimedia documents.

Table 3–13: Asset result variables

Name	Description
asset_Category	The data type category of the asset
asset_MIME-Type	The MIME type associated with the asset category
asset_Title	The title for the asset
asset_Subject	The subject of the asset description
asset_Author	The author for the asset
asset_Keywords	Any keywords associated with the asset
asset_url_*trackname*	The URL reference to a VDF track within the VDF metadata structure. For example: ${asset_url_Binary Document} refers to a binary document, if it exists. The ${asset_url_Small Keyframe} variable is the URL to preview images in the small keyframe track. And, the ${asset_url_Large Keyframe} variable is a URL to preview large keyframes, if they exist in the asset. To use these variable, you do not need to perform a fetch command.
asset_IsReference	An indicator for whether an asset exists and is a pointer to a reference. If the asset exists and is a reference, asset_IsReference is set to YES. If the asset is a VDF, the value is not set. The sample templates use a VTL_IF statement to determine whether the asset_IsReference exists. It it exists, the reference filename displays.

Note: If you want to add a category that is not included in the sample templates, you can do so as long as the Verity search engine supports searching on this category data type. To do so, add a mapping for the category MIME type to the `vss_ftype.xml` file. For the category to appear on the search pages, modify the `vss.js` fils as well to include the new category in the choices on the sample templates.

Result arrays

The CGI program stores resulting data, such as information on each record label, proxy information, within array structures. To access, and display in a template, the data within an array, use the `VTL_LOOP` tag to loop through each element in the array. Some arrays can contain inner arrays. For example, if you want to display a set of results and the corresponding highlighted text for each match within the closed-caption text for each result, include in a result template the following `VTL_LOOP` statements:

```
<VTL_LOOP NAME="results">
  <VTL_LOOP NAME="text_Closed Caption">
    <VTL_LOOP NAME="hitext">${plain}<b class="hits">${highlight}</b>
    </VTL_LOOP><br>
  </VTL_LOOP>
</VTL_LOOP>
```

In this example, the CGI program loops through the closed-caption text for each result and renders hits, or query matches, in bold.

Table 3–14 describes the arrays you can use in a template to loop through and display results. The `results` array can contain any other arrays, except for the `pagenav` array, which is for managing page bars.

Table 3–14: result arrays

Array	Description	Reference
clabels and vlabels	Contains information on asset labels	page 105

Table 3–14: result arrays (*Continued*)

Array	Description	Reference
cschema and vschema	Contains information on the schema that the asset file uses	page 106
pagenav	Contains information on the page navigation	page 108
proxies	Contains information on the proxy for the asset	page 107
results	Contains information on each search result	page 97
text	Information on each text track that is returned is named with the track name appended the prefix `text_`. For example, information on the closed-caption track is contained in an array named `text_Closed Caption` (with the space between the word Closed and Caption).	page 109
hitext	Inside each text array, this inner array loops over the highlighted and unhighlighted text of terms that match the highlight criteria.	page 110
video_results array	This array contains an entry for each unique asset ID. Each entry contains a results array with all results that correspond to this asset.	page 110

The clabels and vlabels result arrays

All labels are indexed in the array `clabels`, for clip labels, and the array `vlabels`, for video labels. Other asset types also contain these labels once

the asset is uploaded into VSS and it is therefore contained within a VDF file. Table 3–15 describes each variable within these label arrays.

Table 3–15: Array result variables

Name	Description
label_name	Name of the label. For example, `clabel_Description`
label_display_name	Display name of the label, after removing `clabel_` or `vlabel_`. For example, `Description`
label_value	Value of the label
label schema information	If the variable `xml_get_schema` is set, schema information for this label will also be added. See cschema and vschema.

The following example illustrates how you can use these array variables to loop through an array to display a label name for each result.

```
<vtl_loop name="clabels”>
<br>
<b>${label_display_name}:</b> ${label_value}<br>
</vtl_loop>
```

The cschema and vschema result arrays

If the variable `xml_get_schema` is set, each result will have clip and video schema information stored in arrays called `cschema` (for clip schema) and `vschema` (for video schema). The schema information is also available in the `clabels` and `vlabels` arrays, but these arrays are more useful when adding a new clip to a video. Table 3–16 lists the variables that each array includes

Table 3–16: Schema and vschema array variables

Name	Description
label_name	The name of the label. For example, `clabel_Description`

Table 3–16: Schema and vschema array variables (*Continued*)

Name	Description
label_display_name	The display name of the label, after removing `clabel_` or `vlabel_`. For example, `Description`
label_required	If the label is required: `yes`, otherwise `no`
label_type	The type of the label: `date`, `string`, `int`, or `float`
label_width	The width of the label
label_input_method	The input method: `entry`, `single_select`, `multi_select`, or `edit_select`
label_select_list	Array that contains a list of the items in the select list if the input method is `single_select`, `multi_select`, or `edit_select`. Each item has the variable name `label_select_list_item`

Proxy result array

Information for each proxy is listed in the `proxies` array, which contains the same data as the default proxy. Table 3–17 lists the proxy result array variables.

Table 3–17: Proxy information results

Name	Description
proxy_bitrate	Proxy bitrate
proxy_framerate	Proxy frame rate
proxy_height	Proxy height

Table 3–17: Proxy information results (*Continued*)

Name	Description
proxy_mime_type	Proxy MIME type
proxy_offset	Proxy offset
proxy_url	Proxy URL
proxy_width	Proxy width
proxy_is_default	1 if this is the default proxy, otherwise undefined

Page navigation result array

Use the `pagenav` array to manage which results to display on each page. For example, if you want to create a page navigation bar, you can use the URL information stored in the `pagenav` array to determine which query results to display on each page. The following `VTL_LOOP` statement (which is used in the sample `results_footer.tmpl` template in the default `vs/conf/accounts/sample/search/` directory) demonstrates a page bar.

```
<VTL_LOOP NAME="pagenav">
   <VTL_IF NAME="current">
      <VTL_ELSE>
 <b><a href=
"${nav_search_uri}/${account}/${view}?${page_query}">
   </VTL_IF>
     ${page}<VTL_IF NAME="current"></b><VTL_ELSE></a></VTL_IF>
</VTL_LOOP>
```

In this example, the URL uses the variable, `nav_search_uri`, that is defined in the view configuration file.

The `pagenav` array can have as many members as defined in the query control variable max_pages. Table 3–18 lists the `pagenav` variables for each page.

See also next_page_query and previous_page_ query.

Table 3–18: URL information results

Name	Description
page	Page number
page_query	Query parameters to go to that page
current	1 if this is the current page

Text result arrays

Each text track returned by a search is in an array `text_` added to the track name. A common array is `text_Closed Caption` (with the space). Table 3–19 lists the elements that are contained in each entry in the text arrays.

Table 3–19: Text array results

Name	Description
hit_line	If a highlight hit was found in this text record, this number is defined by increments of one.
text	Text record
timein	In-time in SMPTE
timeout	Out-time in SMPTE
timein_msec	In-time in msec.
timeout_msec	Out-time in msec
weight	Weight value for weighted text tracks

The hitext result array

Within the text record array, a smaller `hitext` array loops over the highlighted and unhighlighted text of any terms matching the highlight criteria.

Table 3–20: Highlight text results

Name	Description
plain	Stretch of non-matching words
highlight	Stretch of matching words. Normally you will want to surround this variable by some sort of emphasis tag.

The video_results result array

The `video_results` array contains an entry for each unique video asset ID. Within this entry is a results array that contains all result entries that correspond to this video asset.

The `video_results` array may be used to produce a template that organizes clip results by video. A new `video_results` array entry is created each time the video asset ID changes. Thus to be effective, results must be sorted using the video asset as a primary key.

Error handling

If a query returns an error, the variables `error_code` and `error_string` are set with appropriate values. You can use the VTL_IF tag to determine whether or not to display an error page.

Internationalization issues

The character encoding of the templates is determined by the `char_encoding` variable (which defaults to ISO-8859-1). Internally, the CGI search program, vss_SR.exe, works in UTF-8, converting input form variables from the template encoding to the internal character set. All variables

and variable names are represented in this internal character set, and Perl functions are expected to operate in this character set as well. A Virage Perl library is available to convert between ISO-8859-1 and UTF-8. Virage supports conversions to other characters sets, including `KS_C_5601-1987`.

The `vss_SR` CGI program puts the character encoding on the Content-Type HTTP header. For WIN-ANSI (the default character encoding on a Windows system), the character encoding is left blank. In this case, characters are converted to and from the Windows default ANSI code page correctly, but the browser must be set to view the correct character set. Alternatively, template files might contain a meta tag in the head section declaring the character encoding. For example, this meta tag declares a page to use Shift JIS encoding:

```
<meta http-equiv="Content-Type" content="text/html; charset=
Shift_JIS">
```

The account and view names locate the configuration files that in turn hold the character set configuration variables for each template. To avoid difficulties decoding the account and view names before you determine the character set, Virage recommends that these names be entirely in 7-bit ASCII. Note that these names are only used to locate account and view information in the directories, and that all other variables can be expressed in the template encoding.

The locale for date representation is controlled by the `date_locale` variable. Use this only to format date strings with the `:d` variable modifier.

If you are developing templates using non-western characters, see "Working with international templates" on page 69.

Executing vss_SR from the command-line

You can run the CGI programs from the command line for testing purposes. In this mode, the CGI program takes a single argument representing the portion of the URL following the CGI. For example:

```
    vss_SR.exe '/sample/admin?query=movie'
```

executed the `vss_SR.exe` CGI as if it had been invoked as a CGI with the following URL
`/vss-bin/vss_SR.exe/sample/admin?query=movie.`

A The URI Builder Script

This section reviews a helpful JavaScript that can help template designers format the URI part of URLs so that they are easier to read by the human eye. It is not a requirement to use this script for any other templates. It is simply a script that is used in the sample templates to help template designers read complex URIs.

While URIs are generally compact, Virage Solution Server URIs, because of the length and number of query parameters contained within them, are often quite long, difficult to read, and difficult to modify. Because of this, Virage has created a simple, general-purpose, client-side utility that allows web designers to express complex URIs in terms of their component parts. In particular, it allows the complex query component of URIs to be written with name-value pairs on separate lines to make it easier to read the URIs, easier to maintain them, and easier to integrate with Virage Template Language control structures within them.

Basically, the URI_Builder script transforms input from one format to another. This is what these formats look like:

Table 4–1: URI builder formatting

Without URI_builder	With URI_builder
<a href="${nav_search_uri}/${account}/ ${view}?tem- plate=viewer_frameset.tmpl&proxy_mime_typ e=${prox_mime_type}&search_mode=${searc h_mode}&result_type=${result_type}&ass et_id=${asset_id}&video_asset_id=${video_ass et_id}&query=${query:j:h}<VTL_IF NAME="result_type"VALUE="clips">&timein_ msec=${cinfo_timein_msec}&timeout_msec= ${cinfo_timeout_msec}&timein=${cinfo_timei n}&timeout=${cinfo_timeout}</VTL_IF>"> * This URI cannot contain line breaks.	<a href="javascript: location = URI_builder('SCHEME', 'AUTHORITY', 'PATH', '${action_view_uri}', '${account}', '${view}', 'QUERY', 'template=viewer_frameset.tmpl', 'proxy_mime_type=${proxy_mime_type}', 'search_mode=${search_mode}', 'result_type=${result_type}', 'asset_id=${asset_id}', 'video_asset_id=${video_asset_id}', 'query=${query:j:h}', <VTL_IF NAME="result_type" VALUE="clips"> 'timein_msec=${cinfo_timein_msec}', 'timeout_msec=${cinfo_timeout_msec}', 'timein=${cinfo_timein}', 'timeout=${cinfo_timeout}', </VTL_IF> 'FRAGMENT')">

URI structure

The general structure of URIs shows that each one can be broken down into five basic components: a scheme, an authority, a path, a query, and a fragment. Only the path component is required.

The *scheme* identifies the protocol used to send and receive the identified data. Typical schemes are FTP, HTTP, and RTSP.

The *authority* identifies the computer making the content available over the network. An authority is ultimately an IP address, but is more often expressed as a computer name followed by a domain name, such as 'www.virage.com'.

The *path* identifies the file path and file name of the identified content relative to the server root. The server root is dependent on the server being used. The Apache HTTP server typically has a directory called htdocs that it identifies as its root directory. The RealServer, an RTSP server, typically uses a directory called Content for its root.

The *query* component is a series of named parameters along with their respective values that are passed by the content server to an associated CGI application. Within the query, names are separated from values by an equal sign, and name-value pairs are separated from each other by semicolons or ampersands.

The *fragment* component is passed back to the client in hopes that the client can take the viewer to the identified named destination within the identified content. In HTML, fragments are used in conjunction with named anchors, and most web browsers will scroll the page to the identified named location after it loads the requested content.

Each URI component is separated from the other components with a separator unique to its kind. The scheme is separated from the authority by a trailing colon. The authority is separated from the scheme by two forward slashes. If an authority is expressed, then the path always begins with a forward slash representing the root content directory. The query is separated from the path by a question mark. The fragment is separated from the path or the optional query by a pound-sign.

For example, in the URI
`http://www.virage.com/cgi-bin/t.pl?f=john;l=doe#lst`

'http' is the scheme (note the trailing colon separator). The text www.virage.com is the authority (note the leading '`//`' separator). The path is '/cgi-bin/test.pl'. (There is no separator required for the path separator since it's the sole required component.) The query (separated from the path by a question mark) contains two name-value pairs: 'f', which is set to the value 'john', and 'l', which is set to the value 'doe'. The fragment is 'lst', so if the content delivered by 't.pl' is an HTML document that contains a named destination such as `<a name="list"><h1>List</h1></a>`, most browsers will scroll the content so the List heading is in the upper-left corner.

Semicolons or ampersands?

URIs constructed with the URI Builder use semicolons to separate name-value pairs within the query components of the URIs it builds. It is unfortunate that most web browsers separate name-value pairs with ampersands when constructing URIs from form submissions. It encourages web developers to cut and paste form-generated URIs from the address bar in their browser directly into `href` or `src` attributes in HTML documents.

Remember that in both SGML and XML, ampersands mark the beginning of an include statement called an *entity*. Entities have the general form of an ampersand, followed by letters, numbers, or underscores, followed by a semicolon. Note that

```
http://www.virage.com/cgi-bin/t.pl?f=john&l=doe#lst
```

is a valid URI, but that

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <title>Minimal HTML document</title>
</head>
<body>
   <p><a href="http://www.virage.com/cgi-bin/
t.pl?f=john&l=doe#lst">Click Here.</a></p>
</body>
</html>
```

is not valid HTML. The ampersand embedded in the URI is supposed to be interpreted as the beginning of an XML or SGML entity. The reason it still works is because the HTML parser encounters the equal sign, illegal for entities, before encountering a semicolon that would have ended the entity declaration. The parser goes into deal-with-the-error mode and guesses that you wrote '&' when you meant to write '&'. Naturally, the correct way to compose such an HTML document is by either replacing the '&' with '&' or using ';'.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Minimal HTML document</title>
</head>
<body>
```

```
<p><a href="http://www.virage.com/cgi-bin/
t.pl?f=john&amp;l=doe#lst">Click Here.</a></p>
</body>
</html>
```

Using a semicolon in place of ampersands in URIs is not as universally accepted as using the ampersand ('&'), but it is accepted in all modules used by Virage Solution Server. In particular it is accepted by `CGI.pm`, the most-used CGI module for Perl, which, in turn, is used by all the Virage Solution Server Perl CGI programs. The C-based Virage Solution Server CGI, `vss_SR`, also allows semicolons in place of ampersands.

This usage is in keeping with “Hypertext Markup Language 2.0" (RFC 1866) in which Tim Berners-Lee says, in part:

```
8.2.1. The form-urlencoded Media Type

   The default encoding for all forms is 'application/x-www-form-
   urlencoded'. A form data set is represented in this media type as
   follows:

        1. The form field names and values are escaped: space
        characters are replaced by '+', and then reserved characters
        are escaped as per [URL]; that is, non-alphanumeric
        characters are replaced by '%HH', a percent sign and two
        hexadecimal digits representing the ASCII code of the
        character. Line breaks, as in multiline text field values,
        are represented as CR LF pairs, i.e. '%0D%0A'.

        2. The fields are listed in the order they appear in the
        document with the name separated from the value by '=' and
        the pairs separated from each other by '&'. Fields with null
        values may be omitted. In particular, unselected radio
        buttons and checkboxes should not appear in the encoded
        data, but hidden fields with VALUE attributes present
        should.

            NOTE - The URI from a query form submission can be
            used in a normal anchor style hyperlink.
            Unfortunately, the use of the '&' character to
            separate form fields interacts with its use in SGML
            attribute values as an entity reference delimiter.
            For example, the URI 'http://host/?x=1&y=2' must be
            written '<a href="http://host/?x=1&#38;y=2">' or '<a
            href="http://host/?x=1&amp;y=2">'.
```

```
HTTP server implementors, and in particular, CGI
implementors are encouraged to support the use of
';' in place of '&' to save users the trouble of
escaping '&' characters this way.
```

The problem

In Virage Solution Server, template designers communicate indirectly with the server. They compose URIs that send parameters to one of the CGI programs, typically `vss_SR`. These CGI programs, in turn, use the CGI parameters to compose an XML command document, which it sends to the server. The server responds by sending an XML reply document to the CGI program. The CGI program then decodes the XML reply and exposes the results through the template language.

Often the query component of the URIs requesting the CGI program become long and cumbersome, difficult both to read and maintain. As an example, here's a URI that requests information needed to construct an HTML page equivalent to the clip viewer page in the Virage Solution Server sample edit template:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Non-URI Builder example</title>
</head>
<body>
<p><a href="${nav_search_uri}/${account}/${view}?tem-
plate=viewer_frameset\
.tmpl&amp;proxy_mime_type=${proxy_mime_type}&amp;search_mode=
${sear\
ch_mode}&amp;result_type=${result_type}&amp;asset_id=${asset_
id}&am\
p;video_asset_id=${video_asset_id}&amp;query=${query:j:h}<VTL
_IF NA\
ME="result_type"
VALUE="clips">&amp;timein_msec=${cinfo_timein_msec\
}&amp;timeout_msec=${cinfo_timeout_msec}&amp;timein=${cinfo_t
imein}\
&amp;timeout=${cinfo_timeout}</VTL_IF>">Click me</a></p>
</body>
</html>
```

Note that URIs cannot contain spaces or carriage returns. The URI in the HTML document above contains breaks simply to maintain the formatting of this HTML document that you're currently reading. The white space prohibition affects the layout of the Virage Solution Server template pages in the same way.

A solution

The URI Builder is a client-side JavaScript function that uses the power from the client browser to compose these complex URIs from a formatted JavaScript call.

To show the increased legibility of the URI parameters, here is the HTML document shown above reworked to take advantage of the URI Builder JavaScript function:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>URI Builder example</title>
    <script type="text/javascript"
        src="${javascript_uri}/URI_builder.js"></script>
</head>
<body>
<p><a href="javascript: void (location = URI_builder(
'SCHEME',
'AUTHORITY',
'PATH',
  '${action_view_uri}',
  '${account}',
  '${view}',
'QUERY',
  'template=viewer_frameset.tmpl',
  'proxy_mime_type=${proxy_mime_type}',
  'search_mode=${search_mode}',
  'result_type=${result_type}',
  'asset_id=${asset_id}',
  'video_asset_id=${video_asset_id}',
  'query=${query:j:h}',
<VTL_IF NAME="result_type" VALUE="clips">
  'timein_msec=${cinfo_timein_msec}',
  'timeout_msec=${cinfo_timeout_msec}',
  'timein=${cinfo_timein}',
```

```
  'timeout=${cinfo_timeout}',
</VTL_IF>
'FRAGMENT'
))">Click me</a></p>
</body>
</html>
```

The first thing to notice is the script element in the head element of the revised HTML document. This element loads the URI Builder function (`URI_builder.js`) into the document.

The next thing to notice is the href attribute of the anchor element. The href attribute is supposed to identify a valid URI. Instead it identifies a pseudo-URI beginning with the 'javascript' scheme. This construction is less than ideal, but necessary with the current generation of browsers. Ideally, we would leave off the href attribute entirely and use the onclick attribute instead (without the 'javascript:'), but this doesn't work in current browsers. In practice, the JavaScript in the href attribute is evaluated and the results effectively replace the JavaScript call. Since the return value of URI builder is a valid URI, the browsers use the URI generated by URI Builder as if it was typed directly into the href attribute, just as in the problem example from the previous section.

URI Builder works by sampling the passed data in the order passed. When it encounters one of the special marker parameters (SCHEME, AUTHORITY, PATH, QUERY, FRAGMENT, all uppercase), it interprets subsequent data as data of the type identified until it encounters another special marker parameter.

The URI Builder is designed to work as a perfect replacement for anything that can be a identified in an anchor href attribute. If it doesn't get a SCHEME or AUTHORITY special marker parameter, it assumes parameters are part of the PATH.

We prefer to include all five special parameter markers in each call to URI Builder, even though it is technically not necessary, just for consistency. In other words, if there is no SCHEME and no AUTHORITY, then there is no need for SCHEME, AUTHORITY, or PATH. (There is no need for PATH, since it's the default parameter.)

If you include the colon separator after your scheme, URI Builder will not add a second one.

If you add the '//' separator before your authority, URI Builder will not prepend an additional '//' separator.

All parameters interpreted as part of the PATH are separated from each other by forward slashes.

All parameters interpreted as part of the QUERY are separated from each other by semicolons.

If you include the pound-sign separator before your fragment, URI Builder will not add a second one.

Remember that all parameters in URI Builder are part of a JavaScript function call. They therefore must follow all the JavaScript rules required by JavaScript function parameters. For example, each parameter must be separated from the others with a comma, but there must not be a comma after the last parameter. Also, all text in the parameters must be properly JavaScript-escaped. In particular, certain special ASCII characters, and all non-ASCII characters, must be escaped. Consider using the :j VTL template variable modifier on all variables that you use in URI Builder calls.

Names and values in the query are URI-encoded before the URI is returned. Do not pre-URI-encode your values when you compose your URI Builder calls. In other words, if your parameter is 't=this one', do not enter it as 't=this+one' or 't=this%20one' since URI builder will do this escaping for you.

At some point, we might add additional tokens to help with URI creation, such as DOMAIN and PORT as subcategories of AUTHORITY, or PATH_INFO as a subcategory of PATH, but none of these are currently implemented.

B SMILClock

Template designers can use this JavaScript to modify time formats.

Synopsis

```
var inTime = new SMILClock("smpte-25=01:14:33:22");
```

Description

SMILClock reads and writes any SMIL 1.0-compliant time format, possibly changing formats before writing.

The SMIL-compliant formats are described in the SMIL Recommendation in the descriptions of the attributes “clock value” (in section 4.2.1) and "clip-begin" (in section 4.2.3).

For convenience, the relevant information is repeated here.

clock value

Clock values have the following syntax:

```
Clock-val          ::= Full-clock-val | Partial-clock-val |
Timecount-val
Full-clock-val      ::= Hours ":" Minutes ":" Seconds ("."
Fraction)?
Partial-clock-val   ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val       ::= Timecount ("." Fraction)?
                       ("h" | "min" | "s" | "ms")? ; default
is "s"
Hours               ::= 2DIGIT; any positive number
Minutes             ::= 2DIGIT; range from 00 to 59
```

```
Seconds              ::= 2DIGIT; range from 00 to 59
Fraction             ::= DIGIT+
Timecount            ::= DIGIT+
2DIGIT               ::= DIGIT DIGIT
DIGIT                ::= [0-9]
```

The following are examples of legal clock values:

- Full clock value: 02:30:03 = 2 hours, 30 minutes and 3 seconds
- Partial clock value: 02:33 = 2 minutes and 33 seconds
- Timecount values:
  3h = 3 hours
  45min = 45 minutes
  30s = 30 seconds
  5ms = 5 milliseconds

A fraction x with n digits represents the following value:
x * 1/10**n

Examples:
00.5s = 5 * 1/10 seconds = 500 milliseconds
00:00.005 = 5 * 1/1000 seconds = 5 milliseconds

clip-begin

The clip-begin attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object.

Values in the clip-begin attribute have the following syntax:

```
Clip-time-value   ::= Metric "=" ( Clock-val | Smpte-val )
Metric            ::= Smpte-type | "npt"
Smpte-type        ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val         ::= Hours ":" Minutes ":" Seconds
                      [ ":" Frames [ "." Subframes ]]
Hours             ::= 2DIGIT
Minutes           ::= 2DIGIT
Seconds           ::= 2DIGIT
Frames            ::= 2DIGIT
Subframes         ::= 2DIGIT
```

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

SMPTE time stamp

SMPTE timecodes can be used for frame-level access accuracy. The metric specifier can have the following values:

smpte
smpte-30-drop

These values indicate the use of the "SMPTE 30 drop" format with 29.97 frames per second. The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.

smpte-25

The "frames" field in the time specification can assume the values 0 through 24.

The time value has the format hours:minutes:seconds:frames.subframes. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth of a frame.

Examples:

```
clip-begin="smpte=10:12:33:20"
```

Normal play time

Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "npt", and the syntax of the time value is identical to the syntax of SMIL clock values.

Examples:

```
clip-begin="npt=123.45s"
clip-begin="npt=12:05:35.3"
```

Constructor

The constructor is "SMILTime" and it takes a single time value, for example:

```
var outTime = new SMILClock("5min");
```

Properties

Class Properties

`SMILClock.VERSION`

The version of the code used to define the SMILClock object.

Instance Properties

SMILClock.`ms`

The SMILClock object tracks data in milliseconds. Do not read or write the property directly. Use the defined methods to read the value. Create a new object if you want to write the value.

Methods

Class Methods

`SMILClock.parse(`*time*`)`

Accepts time values in a variety of formats and returns the equivalent milliseconds as an integer.

`SMILClock.getBegin(`*time, time*`)`

Pass this a duration and an end time and it returns a SMILClock object representing the appropriate begin time.

`SMILClock.getDur(`*time, time*`)`

Pass this a start time and an end time and it returns a SMILClock object representing the appropriate duration.

`SMILClock.getEnd(`*time, time*`)`

Pass this a start time and a duration and it returns a SMILClock object representing the appropriate end time.

`SMILClock.set(time)`

Takes a time value and returns a SMILTime object. You can use this as an inline alternative to the constructor.

Instance Methods

`SMILClock.toSMPTE()`

Returns a string showing the SMILClock object in SMPTE-30 format.

`SMILClock.toSMPTE30Drop()`

Returns a string showing the SMILClock object in drop-frame SMPTE-30 format.

`SMILClock.toSMPTE25()`

Returns a string showing the SMILClock object in SMPTE-25 format.

`SMILClock.toH()`

Returns a floating point number representing the SMILClock object in hours rounded to two decimal places.

`SMILClock.toMIN()`

Returns a floating point number representing the SMILClock object in minutes rounded to two decimal places.

`SMILClock.toS()`

Returns a floating point number representing the SMILClock object in seconds rounded to two decimal places.

`SMILClock.toMS()`

Returns an integer representing the SMILClock object in milliseconds.

SMILClock.`toHMS()`

Returns a string showing the SMILClock object in HH:MM:SS.xx format. The value is always four digits: hours, colon, minutes, colon, seconds, period, centiseconds.

SMILClock.`toh()`

Returns a string showing the SMILClock object in hours rounded to two decimal places. Has an 'h' appended.

SMILClock.`tomin()`

Returns a string showing the SMILClock object in minutes rounded to two decimal places. Has 'min' appended.

SMILClock.`tos()`

Returns a string showing the SMILClock object in seconds rounded to two decimal places. Has an 's' appended.

SMILClock.`toms()`

Returns a string showing the SMILClock object in milliseconds. Has 'ms' appended.

SMILClock.`tohms()`

Returns a string showing the SMILClock object in HH:MM:SS.xx format. If the value is less than an hour, then the HH: is omitted. If the value is less than a minute, then the HH:MM: is omitted and an 's' is appended.

SMILClock.`toValue()`

Overrides standard `toValue` method to be identical to `toMS`. Returns an integer representing the SMILClock object in milliseconds.

SMILClock.`toString()`

Overrides standard `toString` method to be identical to `tohms`. Returns a string or a floating point number showing the SMILClock object in HH:MM:SS format.

Examples

Here's a typical use. See the individual method documentation for more options and information.

```
<script type="text/javascript"><!--
document.write(SMILClock.set("4588933ms").toSMPTE());
// --></script>
```

This writes `01:16:28:28` into the HTML document.

Files

The SMILClock class is defined in a stand-alone JavaScript file, SMIL-Clock.js. Add the following line to the head element of your HTML file to make the class available to your scripts.

```
<script type="text/javascript" src="SMILClock.js"></script>
```

The src attribute contains a URI, so this example assumes that SMIL-Clock.js in the same directory as the HTML file.

The SMILClock.js is located in the `vs/vssdocs/accounts/sample/edit/SMILClock.js` directory. The file will not display in your browser. To download it, right-click the link if you're using Windows, and then click Save Link As in the menu that appears.

See also

See also the SMIL Recommendation.

Caveats

Don't forget to put the empty parentheses after all methods. If you forget, you get JavaScript code rather than the time value you were expecting.

Note that SMPTE values are timecodes, not time values. SMPTE timecodes represent frames in a video and are useful for identifying a beginnings and ends of video segments for the purpose of annotating and editing video. SMPTE cannot logically be used to represent a time duration of any sort. For durations, use HH:MM:SS format.

Bugs

Uses JavaScript 1.2 (JScript 3.0) features (such as function literals and RegExp), and is therefore not ECMA-262-2 compliant.

Accepts more time formats than SMIL specifies. Does not complain about most of them. Do not become dependent on them.

The conversion to "smpte-30-drop" is probably not accurate.

C Template File Dependencies

Virage Solution Server ships with a set sample template files for the user interface. Table 6–1 lists each template file in the user-interface along with the corresponding frameset, included template files (included with a `VTL_INCLUDE` statement), JavaScript, and CSS files. If a page in the interface is composed solely of one template file, the URL in the browser displays the name of the template file. Otherwise, use this reference to see which files are embedded within other files. With this information, you can determine file dependencies and assess the impact across files for modifications to any particular file.

You can view the template files for the sample administration view in the `conf/accounts/admin/admin_view/` directory and the `conf/accounts/install_test/search/` directory and the sample view templates in the `conf/accounts/sample/` directory. A couple of the Test pages use static HTML pages and others are rendered from CGI scripts without a template.

If you want to edit the sample views, make modification within the `conf/` directory locations. However, if you want to make modifications that take effect for all views that are created subsequently that are based on the sample views, modify the master templates located in the `master_templates` subdirectory. When you create a new account or view, the solution server copies the sample templates from the `master_templates` subdirectory to new directories where you can customize them to suit your content needs. If you modify the template file, make the appropriate changes to the corresponding HTML online help file.

Table 6–1: Template composition files

Template	VTL includes	JavaScript includes	CSS includes	Frames
Search view				
asx.tmpl	none	none	none	none
email.tmpl	none	vss.js	none	none
email_success.tmpl	none	none	none	none
error.tmpl	none	none	none	none
head.tmpl	none	vss.js, URI_builder.js	${stylesheet_uri}/ vss.css	none
playprefs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
ram.tmpl	none	none	none	none
redirect.tmpl (Based on cookie information, this page redirects to playprefs.tmpl, viewer_mplayer.tmpl, or viewer_realplayer.tmpl)	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
results_footer.tmpl	none	none	none	none
results_header.tmpl	none	none	none	none
results_result.tmpl	none	none	none	none
search.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
smil.tmpl	none	none	none	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
test_install.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl and /vs/accounts/admin/vss.css	none
viewer_cc.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_cc_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs.tmpl, viewer_cc_search.tmpl, viewer_cc.tmpl
viewer_cc_search.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_clips.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_clips_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs.tmpl, viewer_clips.tmpl
viewer_frameset.tmpl (The frames rendered by VTL depend on the kind of assets that play)	head.tmpl	files from head.tmpl	vss.css from head.tmpl	redirect.tmpl, results.tmpl, viewer_cc_fs.tmpl
viewer_incl_mplayer.tmpl	none	none	none	none
viewer_incl_realplayer.tmpl	none	none	none	none
viewer_mplayer.tmpl	head.tmpl, viewer_incl_mplayer.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_realplayer.tmpl	head.tmpl, viewer_incl_realplayer.tmpl	files from head.tmpl	vss.css from head.tmpl	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
viewer_story_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs.tmpl, viewer_story board.tmpl
viewer_storyboard.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_tabs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
incl_categories.tmpl	none	none	none	none
incl_icons_links.tmpl	none	none	none	none
incl_icons_nolinks.tmpl	none	none	none	none
incl_javascript_link_to_viewer.tmpl	none	none	none	none
incl_searchbox_tmpl	none	none	none	none
login.tmpl	head.tmpl incl_results_header.tml	files from head.tmpl	vss.css from head.tmpl	none
results.tmpl	head.tmpl results_header.tmpl results_results.tmpl results_footer.tmpl	files from head.tmpl, plalist.js	vss.css from head.tmpl	none
results_header.tmpl	incl_searchbox.tmpl	none	none	none
results_result.tmpl	incl_javascript_link_to_viewer.tmpl, incl_icons_nolinks.tmpl, incl_icons_links.tmpl incl_categories.tmpl	none	none	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
Edit view (The Edit View contains the same files as the Search View, including the following template files.)				
editor_cc.tmpl	head.tmpl, editor_header.tmpl	files from head.tmpl, SMILClock.js, edit.js, find_replace.js, drift.js	vss.css from head.tmpl	none
editor_clabels.tmpl	head.tmpl, editor_header.tmpl, editor_label.tmpl	files from head.tmpl, vss_form_check.js, SMILClock.js	vss.css from head.tmpl	none
editor_delclip.tmpl	head.tmpl, editor_header.tmpl	files from head.tmpl	vss.css from head.tmpl	none
editor_delvideo.tmpl	head.tmpl, editor_header.tmpl	files from head.tmpl	vss.css from head.tmpl	none
editor_header.tmpl	none	none	none	none
editor_label.tmpl	none	none	none	none
editor_status.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
editor_storyboard.tmpl	head.tmpl, editor_header.tmpl	files from head.tmpl, SMILClock.js, edit.js	vss.css from head.tmpl	none
editor_vlabels.tmpl	head.tmpl, editor_header.tmpl, editor_label.tmpl	files from head.tmpl, vss_form_check.js, SMILClock.js	vss.css from head.tmpl	none
Folders view (The Folders view contains some of the same files in the Search View.)				

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
asx.tmpl	none	none	none	none
cb_create.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_create_confirm.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
cb_delete.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_delete_confirm.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_edit.tmpl	head.tmpl, cb_load_bin.tmpl	files from head.tmpl, cb_core.js, cb_custom.js, cb_ui.js	vss.css from head.tmpl	none
cb_incl_pagebar.tmpl	none	none	none	none
cb_load_bin.tmpl (this file is all JavaScript)	none	none	none	none
cb_login.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_manager.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_page_redirect.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
cb_save_confirm.tmpl	none	none	none	none
cb_search.tmpl	head.tmpl, cb_incl_pagebar.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_setup.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
cb_setup_form.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
cb_setup_redirect.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
cb_share.tmpl	head.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
cb_view_share.tmpl	head.tmpl, cb_incl_pagebar.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
email.tmpl	none	none	none	none
email_success.tmpl	none	none	none	none
error.tmpl	none	none	none	none
head.tmpl	none	vss.js, URI_builder.js	${stylesheet_uri}/ vss.css	none
playprefs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
ram.tmpl	none	none	none	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
redirect.tmpl (Based on cookie information, this page redirects to playprefs.tmpl, viewer_mplayer.tmpl, or viewer_realplayer.tmpl.)	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
results_footer.tmpl	none	none	none	none
results_header.tmpl	none	none	none	none
results_result.tmpl	none	none	none	none
not linked to in Folders pages)	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
smil.tmpl	none	none	none	none
viewer_cc.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_cc_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs.tmpl, viewer_cc_search.tmpl, viewer_cc.tmpl
viewer_cc_search.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_clips.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_clips_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs.tmpl, viewer_clips.tmpl

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
viewer_frameset.tmpl (The frames rendered by VTL depend on the kind of assets that play)	head.tmpl	files from head.tmpl	vss.css from head.tmpl	redirect.tmpl, results.tmpl, viewer_cc_fs. tmpl
viewer_incl_mplayer.tmpl	none	none	none	none
viewer_incl_realplayer.tmpl	none	none	none	none
viewer_mplayer.tmpl	head.tmpl, viewer_incl_mplayer. tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_realplayer.tmpl	head.tmpl, viewer_incl_ realplayer.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_story_fs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	viewer_tabs. tmpl, viewer_story board.tmpl
viewer_storyboard.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
viewer_tabs.tmpl	head.tmpl	files from head.tmpl	vss.css from head.tmpl	none
cb_incl_icons_links.tmpl	none	none	none	none
cb_search.tmpl	head.tmpl cb_incl_pagebar.tmpl cb_incl_icons_links. tmpl incl_categories.tmpl	files from head.tmpl, cb_custom.js	vss.css from head.tmpl	none
email.tmpl	none	vss.js	none	none
incl_categories.tmpl	none	none	none	none

Table 6–1: Template composition files (*Continued*)

Template	VTL includes	JavaScript includes	CSS includes	Frames
login.tmpl	head.tmpl incl_results_header. tml	files from head.tmpl	vss.css from head.tmpl	none
results.tmpl	head.tmpl results_header.tmpl results_results.tmpl results_footer.tmpl	files from head.tmpl, plalist.js	vss.css from head.tmpl	none
results_header.tmpl	incl_searchbox.tmpl	none	none	none
results_result.tmpl	incl_javascript_link_ to_viewer.tmpl, incl_icons_nolinks. tmpl, incl_icons_links.tmpl incl_categories.tmpl	none	none	none

D Template Site Map

This appendix depicts the Virage Solution Server site map for the Administration web pages and the sample account and views. If you customize the templates or add pages to the interface, you can use these tables and flow charts as a guide to organize the application pages.

Table A–1 lists the Index Manager site map.

Table A–1: Index Manager site map

Index Manager site map
Index Manager - List of Indexes
Index Manager - Create Index
Index Manager - Create Index: Choose Schema
Index Manager - Create Index: " " Fields
Index Manager - Create Index Confirmation
Index Manager - Index Properties
Index Manager - Add Asset
Index Manager - Add Asset Confirmation
Index Manager - Asset List
Edit - Asset or ClipLabel Editor
Confirmation of Edits
Index Manager - Replace Asset
Replace Asset Confirmation

Table A–1: Index Manager site map (*Continued*)

Delete
Delete - Confirmation
Export
Index Manager - Delete Index
Index Manager - Delete Index Confirmation
Index Manager - Update Index

Table A–2 lists the Account Manager site map.

Table A–2: Account Manager site map

Account Manager site map
Account Manager - List of Accounts
Account Manager - Create Account
Account Manager - Create Account - View
Account Manager - Create Account Confirmation
Account Manager - Account Properties
Account Manager - Delete Account
Delete Account Confirmation
View Manager - Create View
Create View Confirmation
View Manager - Delete View
Delete View Confirmation

Table A–3 lists the Security pages site map.

Table A–3: Security site map

Security site map
Security - User List
Security - Create User
Security - Delete User (link)
Security - User Details
Security - Change Password
Security - Group Details

Table A–3: Security site map (*Continued*)

- Security - Group List
  - Security - Create Group
  - Security - Delete Group
  - Security - Group Details
    - Security - Manage Group Permissions
    - Security - Manage Group Members
- Security - Permission List
  - Security - Permission Detail
- Security - Configure
  - Security - LDAP Configuration Wizard:
    - Choose LDAP Configuration
    - Enter LDAP Configuration Values
    - Test LDAP Connection
    - Test LDAP Schema
    - Complete VSS LDAP setup
  - Security - LDAP Configuration Details
    - Security - LDAP Configuration Details (edit)
    - Security - Test LDAP Configuration
    - Security - Delete Entry (link)

Table A–4 lists the Folders administration pages site map.

Table A–4: Folders site map

Folders - Master Database Configuration
[CSV or MySQL] Driver Configuration
Confirm Driver Configuration
Server-Wide Configuration Complete
Folders - Select Account
Folders - Folder Management for Account: "[account]"
(Add Users to Account)
(Designate Folder Admins for Account)

Legend

The following icons represent links to pages within the application.

For example, S represents a link to the Search view page.

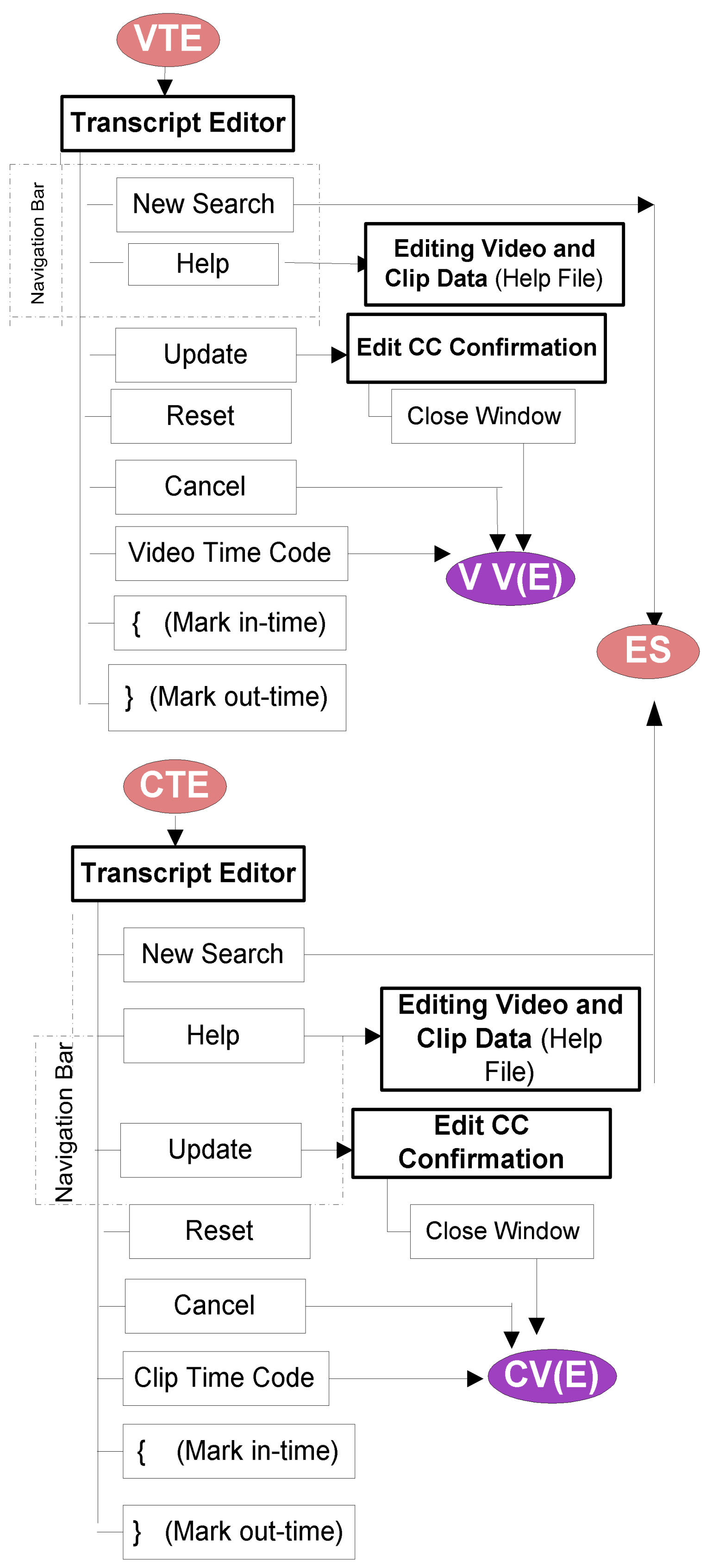
Each group of links are represented in a color. For example, all the links within the Search view page are in yellow and all the links within the Edit view are in red.

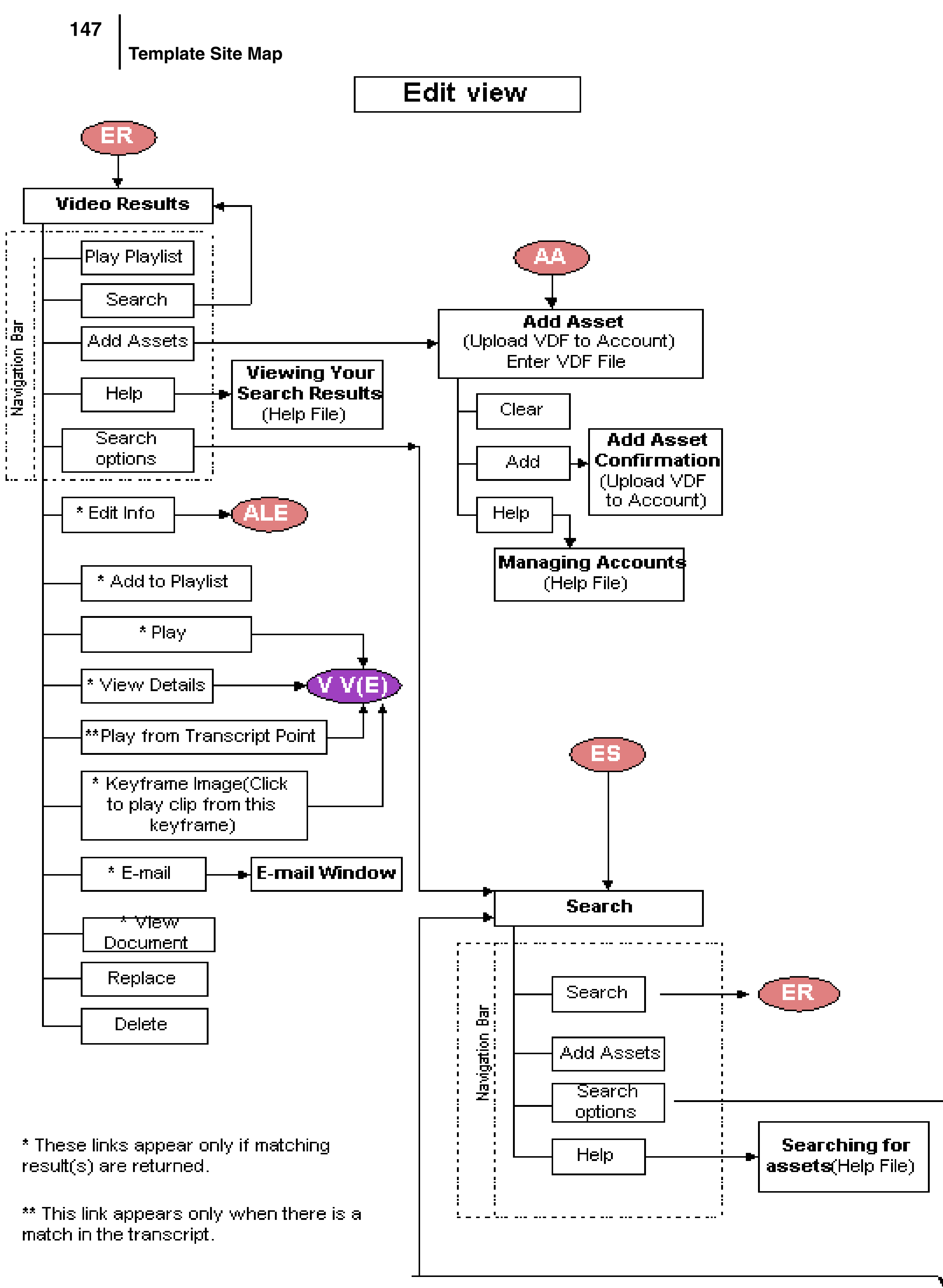
Within the site map illustrations, bold boxes represent individual pages in the user interface.

Other boxes represent links. For example, **Results** is a page in the interface and View Details represents a link within a page.

Icon	Description
S	Search page from the Search view
SR	Results page from the Search view
V V(E)	Video Viewer page from the Edit view
V V(S)	Video Viewer page from the Search view
CV(E)	Clip Viewer page from the Edit view
CV(S)	Clip Viewer page from the Search view
ER	Results page from the Edit view
AA	Add Asset page from the Edit view
ES	Search page from the Edit view
ALE	Asset Label Editor page
CLE	Clip Label Editor page
VTE	Video Transcript Editor page
CTE	Clip Transcript Editor page

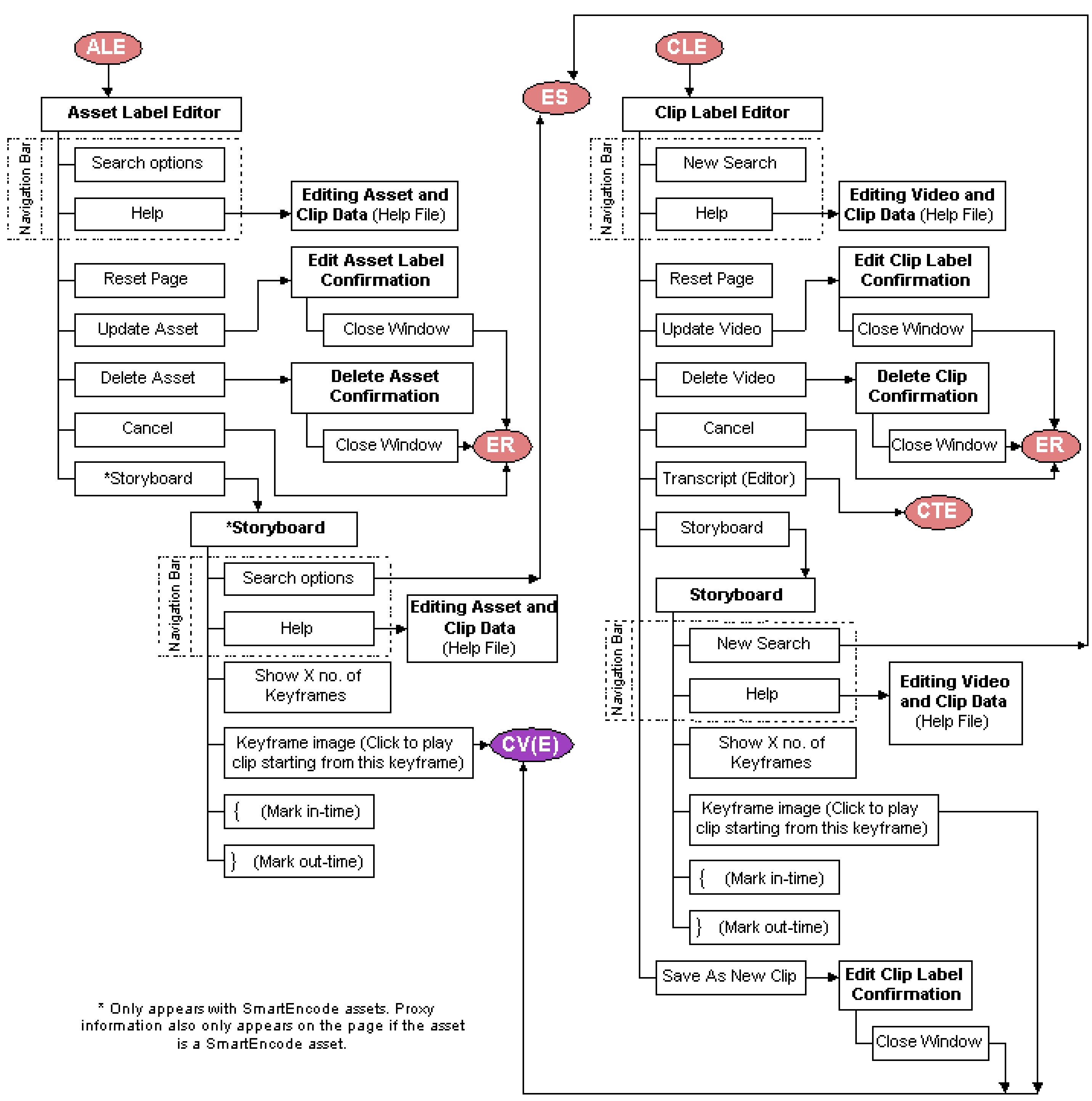
Video and Clip Transcript Editors



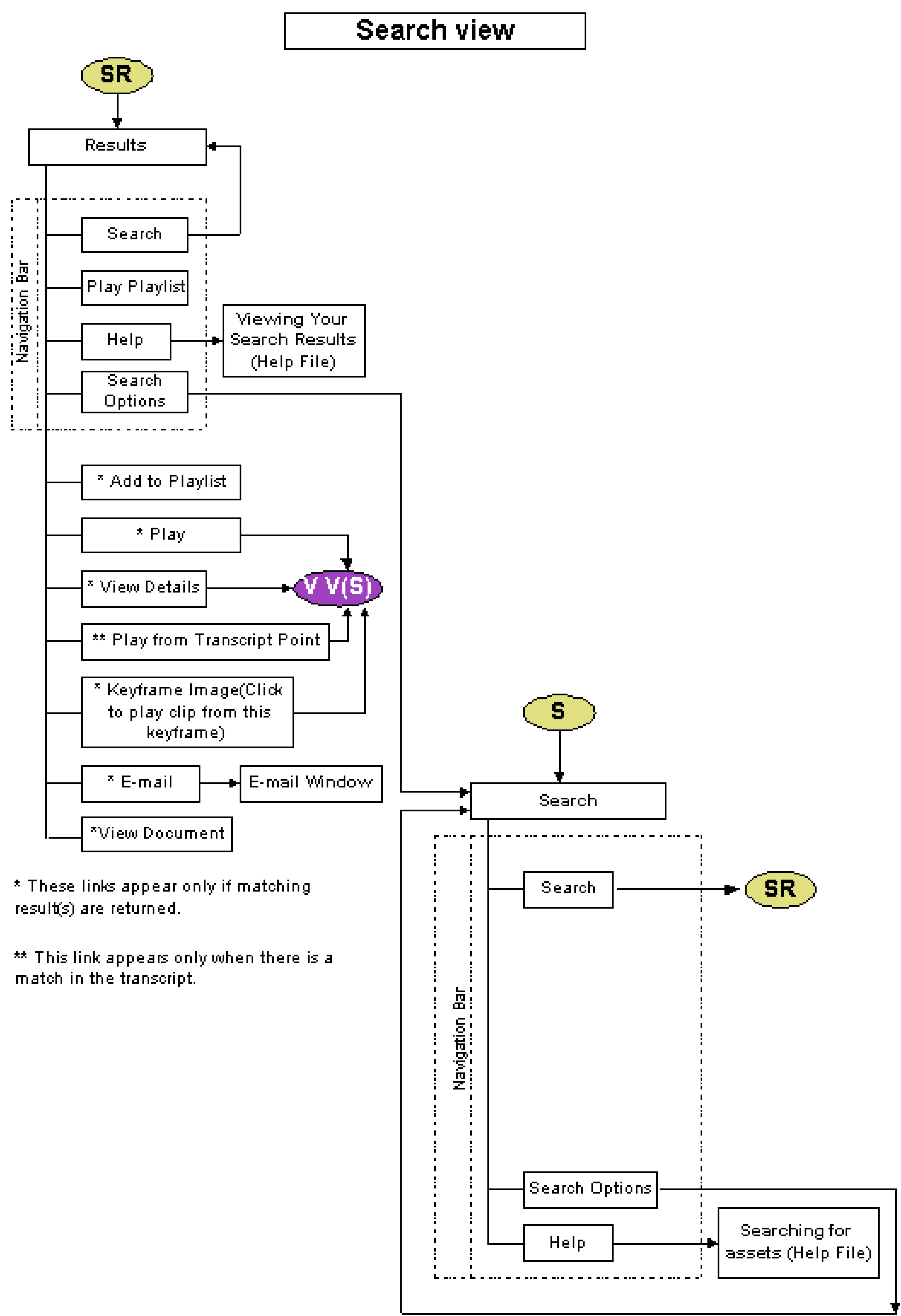


* These links appear only if matching result(s) are returned.

** This link appears only when there is a match in the transcript.



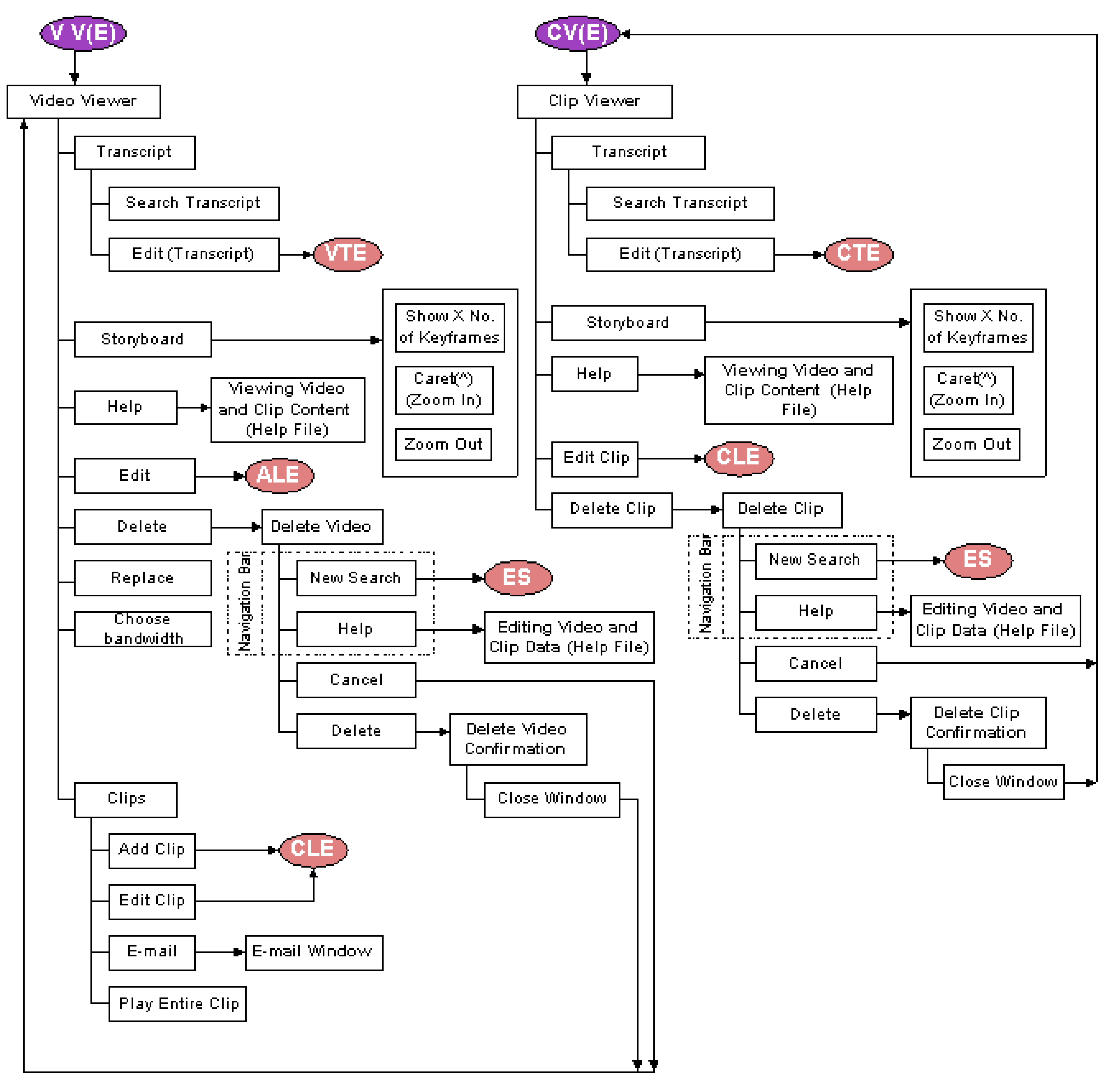
* Only appears with SmartEncode assets. Proxy information also only appears on the page if the asset is a SmartEncode asset.



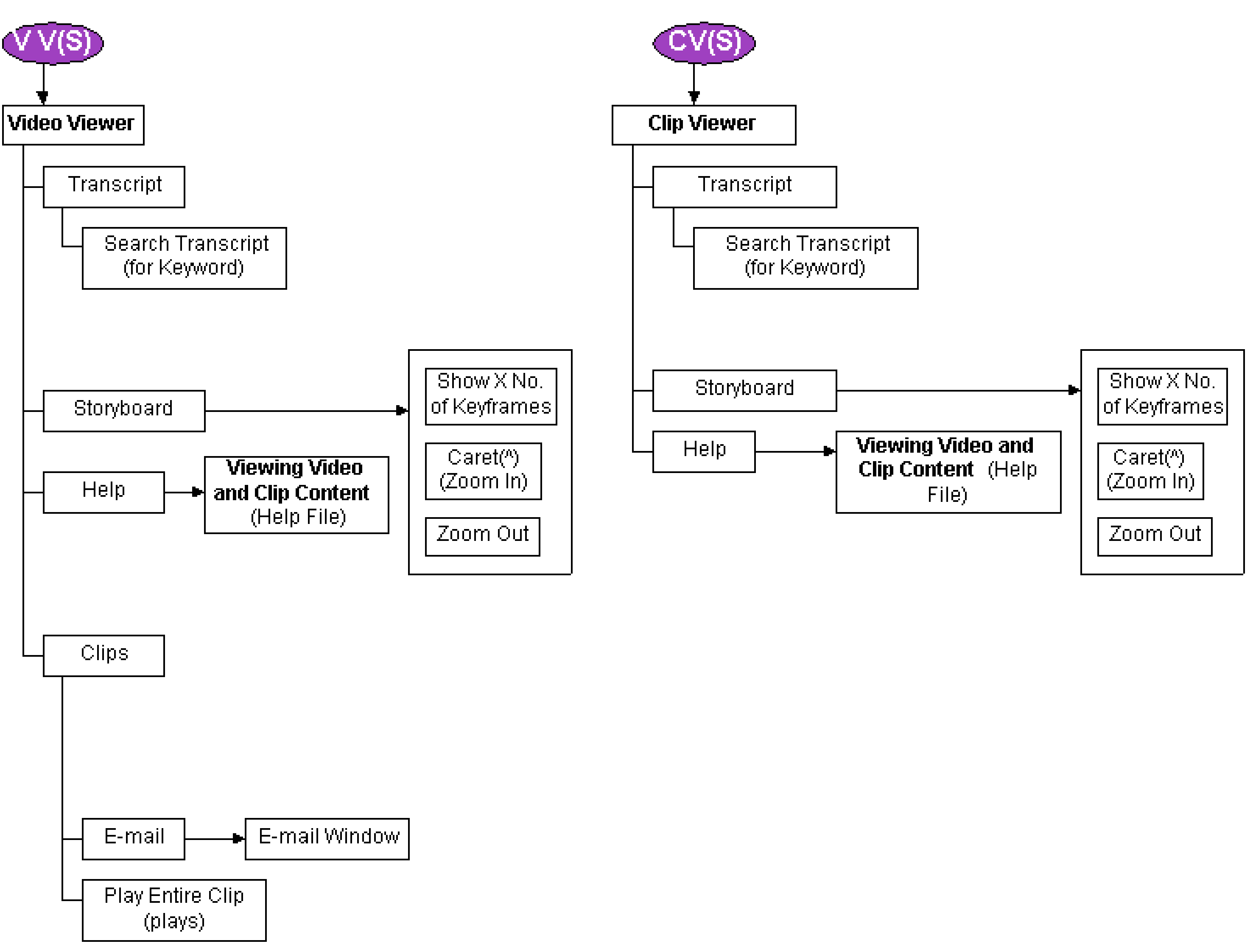
* These links appear only if matching result(s) are returned.

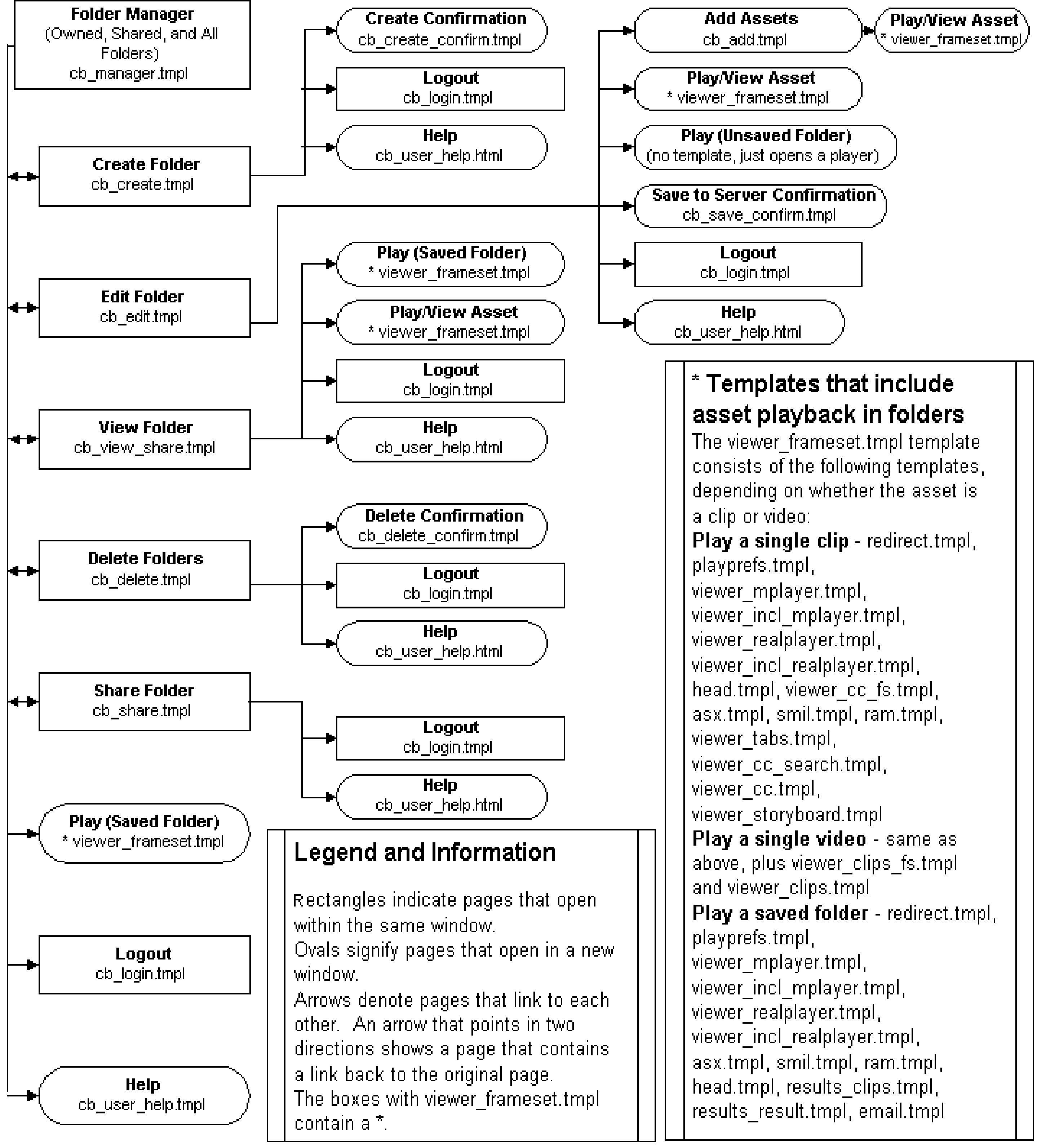
** This link appears only when there is a match in the transcript.

Video and Clip Viewer from the Edit view *



*Video and Clips from SmartEncode asets


Video and Clip Viewer from the Search view
V V(S)
Video Viewer
Transcript
Search Transcript (for Keyword)
Storyboard
Show X No. of Keyframes
Caret(^) (Zoom In)
Zoom Out
Help
Viewing Video and Clip Content (Help File)
Clips
E-mail
E-mail Window
Play Entire Clip (plays)
CV(S)
Clip Viewer
Transcript
Search Transcript (for Keyword)
Storyboard
Help
Viewing Video and Clip Content (Help File)
Show X No. of Keyframes
Caret(^) (Zoom In)
Zoom Out


Folders view
Folder Manager
(Owned, Shared, and All Folders)
cb_manager.tmpl
Create Folder
cb_create.tmpl
Create Confirmation
cb_create_confirm.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.html
Add Assets
cb_add.tmpl
Play/View Asset
* viewer_frameset.tmpl
Play/View Asset
* viewer_frameset.tmpl
Play (Unsaved Folder)
(no template, just opens a player)
Save to Server Confirmation
cb_save_confirm.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.html
Edit Folder
cb_edit.tmpl
Play (Saved Folder)
* viewer_frameset.tmpl
Play/View Asset
* viewer_frameset.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.html
View Folder
cb_view_share.tmpl
Delete Folders
cb_delete.tmpl
Delete Confirmation
cb_delete_confirm.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.html
Share Folder
cb_share.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.html
Play (Saved Folder)
* viewer_frameset.tmpl
Logout
cb_login.tmpl
Help
cb_user_help.tmpl
Legend and Information
Rectangles indicate pages that open within the same window.
Ovals signify pages that open in a new window.
Arrows denote pages that link to each other. An arrow that points in two directions shows a page that contains a link back to the original page.
The boxes with viewer_frameset.tmpl contain a *.
* Templates that include asset playback in folders
The viewer_frameset.tmpl template consists of the following templates, depending on whether the asset is a clip or video:
Play a single clip - redirect.tmpl, playprefs.tmpl, viewer_mplayer.tmpl, viewer_incl_mplayer.tmpl, viewer_realplayer.tmpl, viewer_incl_realplayer.tmpl, head.tmpl, viewer_cc_fs.tmpl, asx.tmpl, smil.tmpl, ram.tmpl, viewer_tabs.tmpl, viewer_cc_search.tmpl, viewer_cc.tmpl, viewer_storyboard.tmpl
Play a single video - same as above, plus viewer_clips_fs.tmpl and viewer_clips.tmpl
Play a saved folder - redirect.tmpl, playprefs.tmpl, viewer_mplayer.tmpl, viewer_incl_mplayer.tmpl, viewer_realplayer.tmpl, viewer_incl_realplayer.tmpl, asx.tmpl, smil.tmpl, ram.tmpl, head.tmpl, results_clips.tmpl, results_result.tmpl, email.tmpl

Glossary

Specific terminology describes the actions, structure, and principles of the Virage Solution Server and associated media. Understanding these terms will help you use Virage Solution Server and this guide.

annotation alphanumeric information associated with video and audio assets generated either manually or automatically. The VideoLogger application automatically generates annotations for closed caption text. A user manually enters annotations that further describe their contents.

ASCII American Standard Code Interchange; a 7-bit standard for encoding alphabetic, numeric, and punctuation characters used in computer systems.

asset ID unique system-generated numeric identification (ID) used to identify assets. Each asset ID is unique within the system: no two assets can share the same asset ID.

asset Assets are any document types. Once uploaded to the VSS, assets are wrapped in VDF files. Metadata, such as text tracks, closed-captioning, and keyframes, are not considered assets.

CC-text closed caption text; ASCII text that is encoded in the original video signal, usually a transcription of spoken dialogue. It's encoded within the vertical blanking interval of an analog video signal. Specialized decoders can extract this text from the signal and provide it to a computer.

clip time delimited segment of video defined simply by an in-time and out-time. Typically, a clip is a short, continuous video segment showing a discrete subject. Clips are defined with in- and out- SMPTE times and are described by VideoLogger with in- and out-keyframes.

clip ID unique system-generated numeric identification (ID) used to identify a clip in the system. Each clip ID is unique within the system: no two clips can share the same clip ID.

clip label identifies the contents of a video clip. A clip label contains in- and out-timecodes that identify a segment of video and the related annotation data entered by a user in predefined fields.

closed-caption track line-21 captioning information from a source video along with in- and out-timecodes that identify a segment of video. In Europe, the closed-caption track contains teletext data.

field definitions individual components of a label, consisting of a set of properties (title, data type, width, input method) and a value. For fields that use the "select" input method, there are "list items." Fields are defined in the `schema` file by a user.

hash hash is a term associated with Perl and other programming. A hash is a data structure. Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. In the case of the solution server, the results hash includes the results of the query on the database, plus any variable interpolation.

keyframes still images generated and time stamped for a specific video asset. Typically, the keyframe is chosen (either manually or automatically) to represent a segment of video. VideoLogger can use an intelligent keyframing algorithm to perform this process automatically. With intelligent keyframing, keyframes represent moments of visual change. With periodic keyframing, keyframes represent moments at regular intervals.

label set of "fields" that describe the contents of a video, audio, or clip asset.

label set definitions for schema labels.

metadata data about data; Since metadata is essentially an index of digital content, metadata is also referred to as a media index.

MPEG-1 Motion Picture Expert Group; video encoding standard defined and accepted by the Motion Picture Expert Group. MPEG-1 video streams may vary in bit rate from a few hundred kilobits per second (kbps) to 1.5 megabits per second (mbps). Image resolution may vary from 160x120 pixels up to 320x240 pixels. Some vendors support extensions to the format for higher bit rates and resolutions.

preview image keyframe that represents the content of a video or clip. VideoLogger by default selects the keyframe closest to the in-time of a video or clip to be the preview image, unless a preview image field is defined in the `schema.xml` file. VideoLogger also allows users to select any keyframe within the video or clip as the preview image. Preview images are also referred to as "beauty frames."

proxy representation of an audio or video asset, usually a low-resolution digital version that can be streamed to desktops. Proxies can technically be any representation of an analog source, including hi-resolution versions, for example: MPEG-2. Proxies provide a user with the capability of viewing or hearing an asset online. Virage uses proxy to mean a digital version of an analog source generated by an encoder. These proxies are typically in MPEG-1, Windows Media, QuickTime, RealVideo, or other compressed video formats.

SMPTE Society of Motion Picture and Television Engineers; standard timecode developed to identify motion-picture frames for editing and broadcasting control. SMPTE is used in video editing to precisely locate frames for editing and audio synchronization. A SMPTE timecode identifies each frame with a code format, hh:mm:ss:ff, denoting hours, minutes, seconds, and frames. For example, a SMPTE timecode of 00:03:29:13 identifies a frame three minutes, 29 seconds, and 13 frames into the video.

storyboard set of keyframes extracted from a segment of video, which visually represent the content. The images in the VideoLogger Keyframe window serve as a storyboard.

text track string of text represented as in-time and out-timecodes that identify a segment of video. Closed captioning is the most common form of a text track; other types can include speaker ID and speech-to-text translation.

timecode counter used to identify an individual frame of a video. The timecode is usually expressed in SMPTE format (hours:min-

utes:seconds:frames or hh:mm:ss:ff). Timecodes can be written on certain tape formats (like Beta SP) as an independent track of information.

VDF Virage Description Format; term used to describe the metadata extracted from a video by Virage logging products.

video asset logical representation of a source video within an asset management system or search and browse application. A video asset is a collection of metadata describing the video, and contains references to representations of the video (for example, a tape, a high-resolution digital version, a low resolution or proxy version.)

video ID unique system-generated numeric identification (ID) used to identify a video in the system. Each video ID is unique within the system or database: no two videos can share the same ID.

video label method used for identifying the entire contents of a video. A user enters annotation data in the video label's predefined fields to describe the contents of the video.

Index

Symbols

${VAR:MODIFIER}. *See* Virage Template Language
${VAR}. *See* Virage Template Language

A

account variables 86
Advanced Search view 15
asset_Author VTL variable 91, 103
asset_Category VTL variable 91, 103
asset_id VTL variable 88, 99
asset_Keywords VTL variable 91, 103
asset_MIME-Type VTL variable 91, 103
asset_Subject VTL variable 91, 103
asset_Title VTL variable 91, 103
asset_url_IsReference VTL variable 103
asset_url_*trackname* VTL variable 103

C

catalog_date VTL variable 99
clabel return arrays 105
clabel_ VTL variable 100
clip information return variables 102
clip label return variables 100
command line syntax 111
conditional statements 79
cschema and vschema return arrays 106
current page VTL variable 109

D

duration VTL variable 102

E

Edit view 15
error handling 110
Export plug-in view 15

F

fetch VTL variable 90
fetch_records VTL variable 90
Folders view 15

H

highlight_string VTL variable 39, 88
hit_line VTL variable 109
hitext result array 110

I

internationalization issues 110

K

keyframe_msec VTL variable 99
keyframe_url VTL variable 99

L

label schema information VTL variable 106
label_display_name VTL variable 106, 107
label_input_method VTL variable 107
label_name VTL variable 106
label_required VTL variable 107
label_select_list VTL variable 107
label_type VTL variable 107
label_value VTL variable 106
label_width VTL variable 107
looping variables 77

M

max_highlights VTL variable 89

max_msec VTL variable 89
max_pages VTL variable 89, 108
max_results VTL variable 89
min_msec VTL variable 89
modifier function 83

N

n_hit_lines VTL variable 99
n_results VTL variable 98
next_page_query VTL variable 98

O

offset VTL variable 87

P

page navigation return arrays 108
page VTL variable 109
page_query VTL variable 109
Perl access functions 83
playlist interstitial 62
Playlist.pm module 57
post_search function 83
pre_search function 83
previous_page_ query VTL variable 98
proxies
  return arrays 107
  return variables 100
proxy offset 23
proxy_bitrate VTL variable 100, 107
proxy_framerate VTL variable 100, 107
proxy_height VTL variable 100, 107
proxy_is_default VTL variable 101, 108
proxy_mime_type VTL variable 100, 108
proxy_offset VTL variable 101, 108
proxy_url VTL variable 101, 108
proxy_width VTL variable 101, 108
ProxySelect.pm module 56

Q

query VTL variable 87
query_array_op VTL variable 88
query_field VTL variable 88
query_op VTL variable 88

R

result_end VTL variable 98
result_start VTL variable 98
result_type VTL variable 99
return arrays. *See* Virage Template Language
return variables 97–102

S

score VTL variable 99
search variables 86
Search view 15
search_type VTL variable 89
sort_dir VTL variable 90
system variables 85

T

template files 78
template VTL variable 90
text return arrays 109
text VTL variable 109
timein VTL variable 102, 109
timein_msec VTL variable 102, 109
timeout VTL variable 102, 109
timeout_msec VTL variable 102, 109
tokens 74
total_results VTL variable 98
Tutorial view 15

U

user.pm 16, 83–85

V

variable stack 81
VDF metadata 22
vdf_filename VTL variable 99
video information return variables 101
video label return variables 100
video results return arrays 110
video_asset_id VTL variable 89, 99
video_results VTL variable 110
view variables 86
vinfo_bitrate VTL variable 101
vinfo_character_encoding VTL variable 101
vinfo_creation_date variable 24

vinfo_creation_date VTL variable 101
vinfo_duration VTL variable 101
vinfo_filepath VTL variable 101
vinfo_fps VTL variable 101
vinfo_height VTL variable 102
vinfo_label_set VTL variable 102
vinfo_modification_date VTL variable 102
vinfo_name VTL variable 102
vinfo_standard VTL variable 102
vinfo_version_major VTL variable 102
vinfo_version_minor VTL variable 102
vinfo_width VTL variable 102
VIR_ALL_DATA 93
VIR_ALL_VDF_DATA 92
VIR_ANY_CLIP_FIELD 92
VIR_ANY_TEXT_TRACK 92
VIR_ANY_VID_FIELD 92
VIR_ASSET_ID_FIELD 91
VIR_CATEGORY_FIELD 93
VIR_CLIP_END_MSEC_FIELD 92
VIR_CLIP_START_MSEC_FIELD 92
VIR_INDEX_DATE 92
VIR_LABELSET_NAME_FIELD 91
VIR_LOG_DATE 92
VIR_RELEVANCE_SCORE 93
VIR_VDF_CREATION_DATE 24, 91, 92
VIR_VDF_FILENAME 93
VIR_VIDEO_ID_FIELD 92
Virage support xi
Virage Template Language 72–111
  conditional statements 79
    VTL_ELSE 79
    VTL_ELSE_IF 79
    VTL_IF 79
  return arrays 105–110
  tokens 74
  user.pm 16, 83–85
  variable syntax 73
  variables
    asset_Author 91, 103
    asset_Category 91, 103
    asset_id 88, 99
    asset_Keywords 91, 103
    asset_MIME-Type 91, 103
    asset_Subject 91, 103
    asset_Title 91, 103
    asset_url_IsReference 103
    asset_url_*trackname* 103
    catalog_date 99
    clabel_ 100
    current page 109
    duration 102
    fetch 90
    fetch_records 90
    highlight_string 39, 88
    hit_line 109
    keyframe_msec 99
    keyframe_url 99
    labe schema information 106
    label_display_name 106, 107
    label_input_method 107
    label_name 106
    label_required 107
    label_select_list 107
    label_type 107
    label_value 106
    label_width 107
    max_highlights 89
    max_msec 89
    max_pages 89, 108
    max_results 89
    min_msec 89
    n_hit_lines 99
    n_results 98
    next_page_query 98
    offset 87
    page 109
    page_query 109
    previous_page_query 98
    proxy_bitrate 100, 107
    proxy_framerate 100, 107

proxy_height 100, 107
proxy_is_default 101, 108
proxy_mime_type 100, 108
proxy_offset 101, 108
proxy_url 101, 108
proxy_width 101, 108
qery_op 88
query 87
query_array_op 88
query_field 88
result_end 98
result_start 98
result_type 99
score 99
search_type 89
sort_dir 90
template 90
text 109
timein 102, 109
timein_msec 102, 109
timeout 102, 109
timeout_msec 102, 109
total_results 98
vdf_filename 99
video_asset_id 89, 99
video_results 110
view variables 86
vinfo_bitrate 101
vinfo_character_encoding 101
vinfo_creation_date 101
vinfo_duration 101
vinfo_filepath 101
vinfo_fps 101
vinfo_height 102
vinfo_label_set 102
vinfo_modification_date 102
vinfo_name 102
vinfo_standard 102
vinfo_version_major 102
vinfo_version_minor 102
vinfo_width 102
vlabel_ 100
weight 109
xml_command 99
xml_debug 91
xml_get_schema 91
xml_reply 99
vlabel_ VTL variable 100
vlabels return arrays 105
VTL. *See* Virage Template Language
VTL_ELSE 79
VTL_ELSE_IF 79
VTL_IF 79
VTL_INCLUDE 78

W

weight VTL variable 109

X

xml_command VTL variable 99
xml_debug VTL variable 91
xml_get_schema VTL variable 91
xml_reply VTL variable 99

Virage® Incorporated
177 Bovet Road, Suite 520 **(650) 573-3210** **www.virage.com**
San Mateo, CA 94402 **(650) 573-3211 fax** **info@virage.com**