eyeQ Bundle Reference Application Guide

Product Version 1.5.0

Published: 30-Jul-2013 15:44

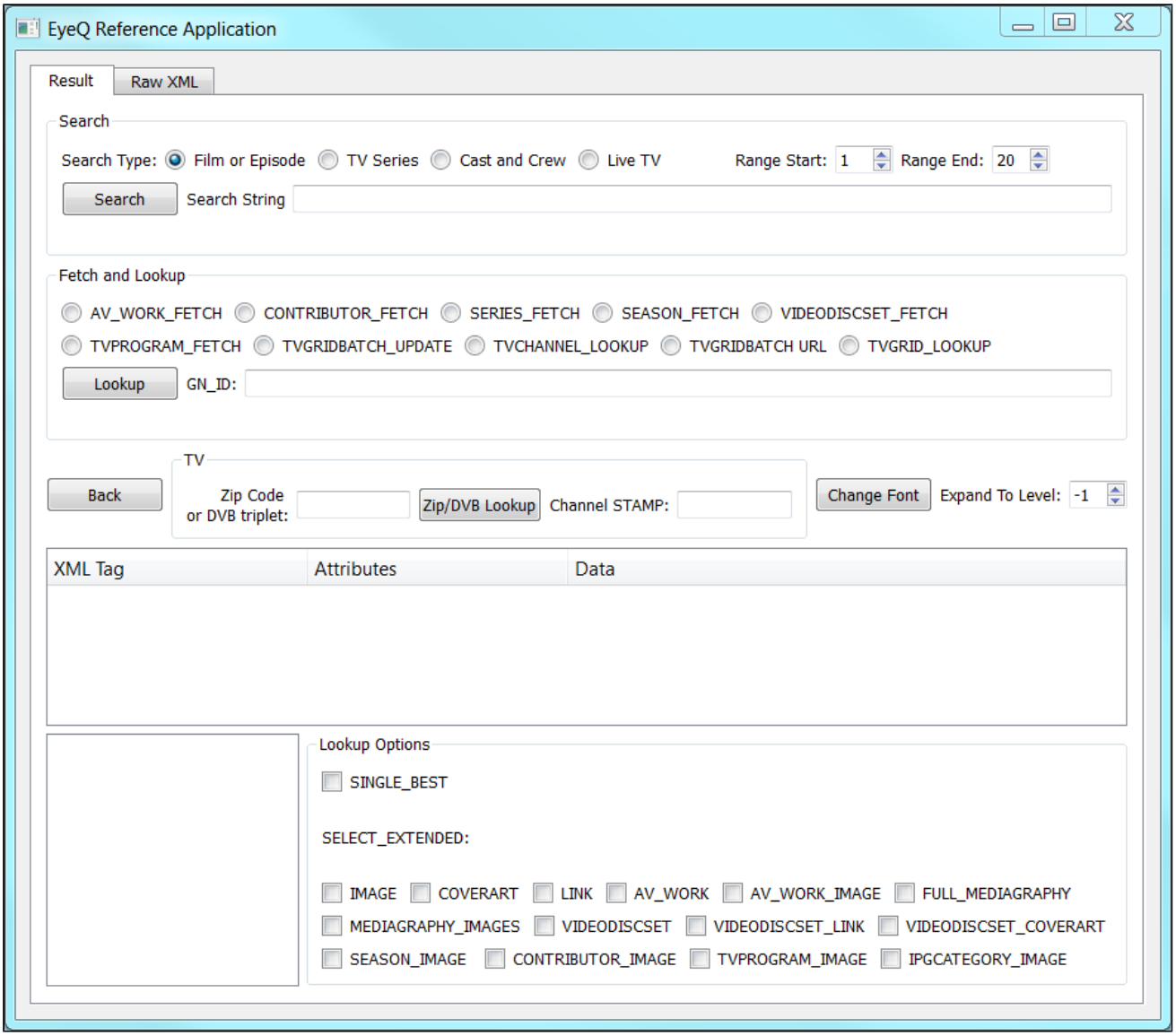
Gracenote, Inc.
2000 Powell Street, Suite 1500
Emeryville, California
94608-1804
www.gracenote.com

Table of Contents

About the Reference Application

The Gracenote eyeQ™ Bundle includes a reference application. This application shows how to do common Video and TV program queries, and process return responses. The UI for this application is basic and not intended for for consumer applications.

The Gracenote eyeQ Reference Application demonstrates how to use the Gracenote eyeQ Web API to access video and interactive TV program content. The application is written in C++ with a simple QT-based UI to show basic queries and responses. *You are not to reuse or re-purpose the source code for this application for consumer applications*. Gracenote provides this code for learning purposes only.



Running the Reference Application

To run the Reference application without installing the Visual Studio development environment:

1. Locate the Reference Application in the eyeQ *binary* distribution, and if needed unzip it.
2. Locate and edit the config.txt file for the following:
    1. Replace the commented-out client_id with the Client ID-Client ID Tag pair or string you received from Gracenote Professional Services and uncomment the line.
    2. Replace the commented-out service_url with the *complete* URL you received from Gracenote Professional Services and uncomment the line.
    3. Optionally, change the country field from its default setting of usa in the config.txt file.
3. Windows specific: Install the Microsoft Visual C++ 2008 Redistributable Package: http://www.microsoft.com/download/en/confirmation.aspx?id=29 (http://www.microsoft.com/downloads/info.aspx?na=41&srcfamilyid=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&srcdisp)
4. Windows specific: Install 32-bit version of OpenSSL, even if you are using Windows 64-bit, from http://www.openssl.org/
5. Locate the Reference Application .exe or .app file and launch it.

Setting up a Gracenote Thin Client

To use eyeQ fingerprinting or TOC lookups, your application will need to use a Gracenote Thin Client. Consult with Gracenote Professional Services as most applications will not utilize the Thin Client. If your application *does not* require fingerprint or TOC lookups, such as a streaming video set top box with no DVD drive, you do not need a Thin Client. For Gracenote eyeQ, the Thin Client does not require a license file, key, or Client ID-Client ID Tag pair.

Setting Up the Reference Application Development Environment

To build and test the Reference application, you will need to set up a specific build environment. Gracenote used Microsoft Visual C++ 2008 and the Nokia QT 4.7.x to create the Reference Application. To build and test the Reference Application code, you should set up a similar environment.
#Install these components:

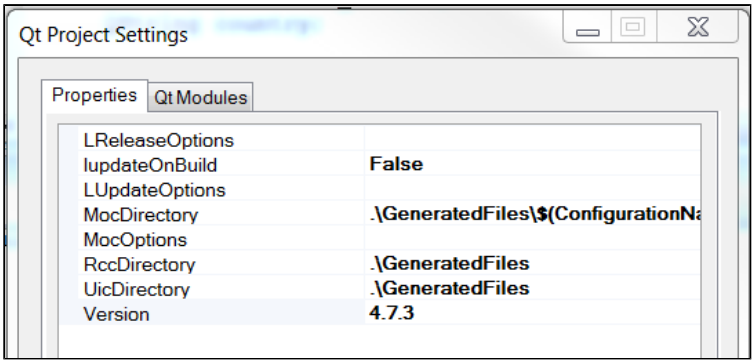
1. Microsoft Visual C++ 2008: http://www.microsoft.com/downloads/en/default.aspx
2. Nokia QT SDK: http://qt.nokia.com/downloads
3. Nokia QT Visual Studio Add-in: http://qt.nokia.com/downloads/visual-studio-add-in
4. OpenSSL for Windows: http://sourceforge.net/projects/gnuwin32/files/openssl/0.9.8h-1/openssl-0.9.8h-1-setup.exe/download
5. Test the installations by building and running the sample applications provided with QT. When these run properly, you can proceed with building and running the eyeQ Reference Application..

Building and Testing the Reference Application

1. Set up and test the development environment. See Setting Up the Reference Application Development Environment.
2. Locate the Reference Application in the eyeQ distribution, and if needed, unzip the folder: eyeq_ref_app_win_n_n_n_n.zip.
3. Locate and edit the config.txt file to add your Client ID string that you got from Gracenote Professional Services. Optionally, change the country field from its default setting of usa in the config.txt file.
4. Replace the service_url value with the Gracenote service URL provided by Gracenote Professional Services.

```
#client_id    9999999-ABCDEF1234567890ABCDEF1234567890
service_url     <gracenote_service_url>
country        usa
# Proxy server support is optional
#proxy_url    your.proxy.server.here.com
#proxy_port    8888
```

5. Open the Visual Studio solution file (.sln) in the Reference application folder. This launches Visual Studio and opens the Reference application.
6. Verify that the QT > QT project settings are set to Version 4.7.3 or higher.



7. Choose Build > Build solution.
8. Run the application, for example Debug > Start Debugging.

Reference Application Query Types

The Reference application demonstrates most of the kinds of queries supported by eyeQ. These generally fall into these categories:

- **Text Search queries**: Text-based searches of the Gracenote Media Service for data about AV Works, TV programs, TV Series, Contributors (Cast and Crew), and other related elements.
- **Object Fetch queries**: Searches the Gracenote Media Service using Gracenote IDs (GN_IDs) for data about about AV Works, TV programs, TV Series, Contributors (Cast and Crew), and other related elements.

- **Video Disc queries**: Searches the Gracenote Media Service for data about the Video disc based on the disc table of contents (TOC). The disc may contain AV Works, TV Programs, or a combination. Results may include data about Series, Seasons, Contributors, and other related elements.
- **Live TV (IPG) queries**: Searches for *instances of* and the data associated with TV Programs and AV Works *airing on TV*. These queries are also known as EPG (electronic program guide) or IPG (interactive program guide) queries.

Using the Reference Application

The following sections describe how to use the Reference application to perform basic queries.
Note : For convenience, double-clicking on responses highlighted in yellow is equivalent to single-clicking the highlighted response and then clicking Lookup. The instructions below use single-clicking to walk through the steps in more detail.

Viewing Live TV Listings for a Channel

1. In the Search pane, click Live TV (EPG) search type.
2. In the TV pane, type a postal code (zipcode) or DVB triplet
3. In the TV pane, click Zip/DVB Lookup. If using a DVB triplet, skip to step 6.
4. In the Responses pane, click a local TV Provider (highlighted in yellow). This populates the ID box in the Fetch pane with the TV Provider's GN_ID.
5. In the Fetch pane, click Lookup to get the list of TV Channels for this provider.
6. Click TVCHANNEL (highlighted in yellow) for the channel you already know shows the TV program you are looking for. This populates the GN_ID box with the GN_ID of the selected channel, and automatically selects the TVGRID radio button.
7. In the TV Pane, ensure the Start and End Dates are appropriate. If they are left blank, they will default to current time minus one hour to the current time plus on hour. Time is in UTC and follows this format: YYYY-MM-DDTHH:mm, where MM, DD, HH, and mm contain a leading zero if necessary. Example: 2011-08-30T00:27
8. Click Lookup.

Testing a Live TV Text Search

To test a Live TV (EPG) text search:

1. In the Fetch panel, select the TVGRID radio button and enter the GN_ID for a channel. For example, try the Food Network.
2. Click Lookup.
3. Look for a TV Program title airing on that channel, such as Ace of Cakes. Note the time and title.
4. Select Live TV for the search type radio button.
5. Enter the TV Program title you selected in the Search String box.
6. Enter the TV channel ID in the TV Channel GN_ID box.

7. Make sure the Start Date and End Date boxes spans the airing time of the program selected in step 3.
8. Click Search. The results should include the TV Program you previously observed in the EPG data for that channel.

Performing a Text Search for a Film (AV Work) or TV Program (Episode)

This search looks for AV Works or TV Programs that match the text string. As you review the results of each step, you can further refine your search to get the data you need from the Gracenote Media Service. You can also use this process to search for TV Series and Cast and Crew.

1. In the Search pane, choose a Film or Episode search type.
2. Type a search string for a Film or TV program, such as *The Simpsons*.
3. In the Query Options area, check the kinds of data you want to get. Note that these checkboxes apply the SELECT_EXTENDED query options. The Reference application does not validate which options are valid for a query. Any options chosen that do not apply to the query are ignored by the Gracenote Media Service.
4. In the Search pane, click Search. The application displays the results in the Responses pane. The initial query returns all matches for the search string.
5. Locate the instance you want to information about, such as the AV Work *The Simpsons Movie*. Click the AV Work. This populates the ID box in the Fetch pane with the GN_ID of the AV Work.
6. In the Fetch pane, click Lookup. This populates the responses pane with detailed information about the AV Work.
7. Using these results, you can get additional information by selecting the highlighted elements (such as Contributor) and fetching their data by clicking Lookup. See the Gracenote eyeQ Implementation Guide for information about the supported elements.

Performing a Text Search for a TV Program from a TV Provider

1. In the Search pane, click Live TV search type.
2. In the TV pane, type a postal code (zipcode) or DVB triplet.
3. In the TV pane, click Zip/DVB Lookup.
4. In the Responses pane, click a local TV Provider (highlighted in yellow). This populates the ID box in the Fetch pane with the TV Provider's GN_ID.
5. In the Fetch pane, click Lookup to get the list of TV Channels for this provider.
6. Read the responses and locate the GN_IDs of each TV Channel you want to search.
7. Type the TV Channel IDs in the TVCHANNEL_ID box of the TV pane. For multiple entries, separate each by a comma.
8. In the Search pane, type a search string for a TV program, such as The Simpsons.
9. In the Query Options area, check the kinds of data you want to be returned. Note that these checkboxes apply the SELECT_EXTENDED query options. The Reference application does not validate which options are valid for a query. Any options chosen that do not apply to the query are ignored by the Gracenote Media Service.

10. In the Search pane, click Search. The application displays the results in the Responses pane.

Linking to Related Objects

After retrieving an object, you can search other related object types by linking the object type and field parameter.

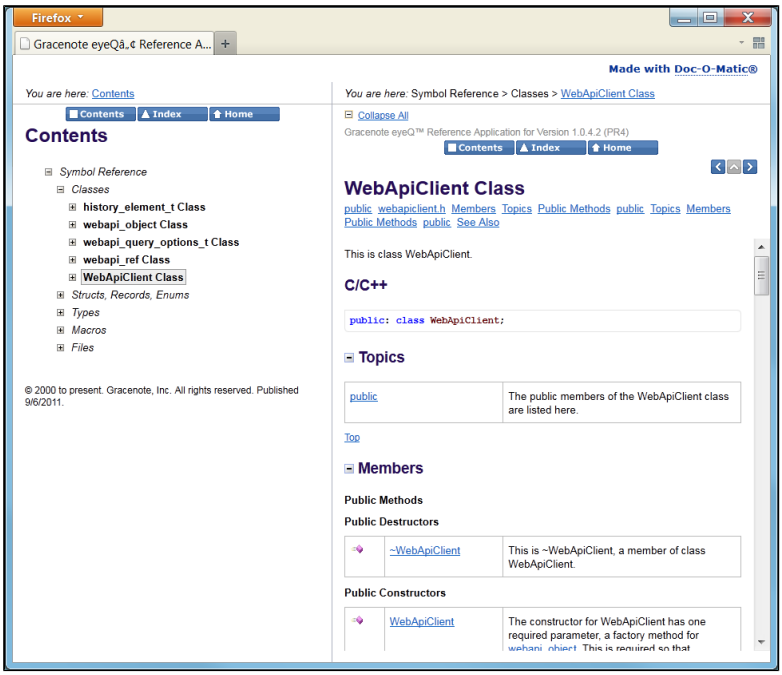
Reference Application Code

The following code listings are excerpted from the Reference application source code. You can find the full source code here in the distribution, or view it in the generated Code Reference documentation.

API Reference Documentation

Gracenote provides Reference Application documentation in HTML format to make it easier to browse and learn about the Reference application. This documentation is similar to other API reference documentation provided with SDKs. It also includes file listings so you can view source code within this documentation. To view the Code Reference documentation:

1. Locate the documentation here: \docs\external\gn_eyeq_reference_app_code_reference\
2. Click on the start_here.html file to launch the documentation.



Reference Application Code Overview

The two primary classes in the Reference application are webapi_ref and webapiclient. The webapi_ref class sets up and manages the UI of the application. It manages network access, image downloads, and general operation of the UI. It also implements higher-level methods that construct and perform XML queries, such as text searches and fetches by GN_ID.

The webapiclient class provides additional, lower-level methods to perform Gracenote eyeQ queries and process responses returned from the Gracenote Media Service.
The listings below highlights some of the key types, structs, macros, and methods defined webapi_ref and webapiclient classes.

Important webapi_ref Methods

The webapi_ref class defines methods that construct and issue high-level queries. Some of the most important ones are:

webapi_ref::lookup_id	Fetches data elements based on their GN_IDs.
webapi_ref::lookup_zip	Looks up the TV providers for the selected postal code (North America) or DVB (Digital Video Broadcast) triplet (Europe). Split the DVB triplet and extract the ONID (Original Network ID).
webapi_ref::text_search	Performs a text search of the text entered into the search field. by the user. Different types of search requests are created depending on which search type radio button the user clicks.
webapi_ref::generate_select_extended	Generates SELECT_EXTENDED string based on currently checked checkboxes in the Reference application. This method adds the list of extended query options to the XML request structure.

Important webapiclient Methods

WebApiClient::fetch	Retrieves a video explore object by its GN_ID
WebApiClient::search	Searches for a video explore object by NAME or TITLE
WebApiClient::toc_lookup	Retrieves a DVD, Blu-ray or CD by disc TOC.
WebApiClient::tv_get_channels	Retrieves a list of TV channels given a TVPROVIDER_ID or ONID.
WebApiClient::tv_get_grid Method	Retrieves EPG data used to build a TV grid.
WebApiClient::tv_get_program	Retrieves a TV program by ID.
WebApiClient::tv_get_program_batch	Retrieves a batch of TV programs.
WebApiClient::tv_get_program_batch_list	Retrieves a list of TV program batches given a list of TV channels
WebApiClient::tv_get_providers	Retrieves a list of TV providers given a postal code.
WebApiClient::tv_search	Searches tv listings for the specified input string.

Types

The application defines four main types:


```
webapi_error_t Type  // The main error return type.
webapi_object_t Type // The main object type.
webapi_object_type_t Enumeration // The enumerated object type.
webapi_tv_setup_t Structure // The main type for TV setup information.
```

For example, the webapi_object_type_t enumeration defines these types.

```
typedef enum {
  webapi_invalid = 0,
  webapi_product,
  webapi_work,
  webapi_series,
  webapi_season,
  webapi_contributor,
  webapi_video_product,
  webapi_fp_fpx,
  webapi_tvproviders,
  webapi_tvchannels,
  webapi_tvbatchlist,
  webapi_tvbatch,
  webapi_tvprogram,
  webapi_tvgrid
} webapi_object_type_t;
```

The webapi_tv_setup_t structure holds the user's current TV setup state, including the postalcode, enabled provider ID, and list of enabled TV channels. Any query that requires access to this state, such as a TV program title search, requires a pointer to one of these structures as an argument.

```
typedef struct webapi_tv_setup_s
{
    QString             postalcode;             // Required for North America
    QString             enabled_provider;       // Required for North America
    QVector<QString>    dvbs;                   // Required for Europe
    QVector<QString>    enabled_channel_ids;    // Channelids enabled for this query
    QVector<QString>    enabled_channel_id_stamps;  // Stamps for each channelid (Optional. If provided,
must be provided for all channels.)
} webapi_tv_setup_t;
```

XML Query Macros

The Reference application defines macros used to construct the XML queries. Some are for programming convenience, others represent eyeQ XML queries. For example, WEBAPI_XML_AV_WORK_FETCH represents the XML query AV_WORK_FETCH. Below are several of the important macro definitions that correspond to Web API queries.


```
#define WEBAPI_XML_AV_WORK_FETCH        "AV_WORK_FETCH"
#define WEBAPI_XML_AV_WORK_SEARCH       "AV_WORK_SEARCH"
#define WEBAPI_XML_CONTRIBUTOR_FETCH    "CONTRIBUTOR_FETCH"
#define WEBAPI_XML_CONTRIBUTOR_SEARCH   "CONTRIBUTOR_SEARCH"
#define WEBAPI_XML_TVCHANNEL_LOOKUP     "TVCHANNEL_LOOKUP"
#define WEBAPI_XML_TVGRID_LOOKUP        "TVGRID_LOOKUP"
#define WEBAPI_XML_TVGRID_SEARCH        "TVGRID_SEARCH"
#define WEBAPI_XML_TVPROGRAM_FETCH      "TVPROGRAM_FETCH"
#define WEBAPI_XML_TVPROGRAM_ID         "TVPROGRAM_ID"
#define WEBAPI_XML_TVPROGRAM_LOOKUP     "TVPROGRAM_LOOKUP"
#define WEBAPI_XML_TVPROGRAM_SEARCH     "TVPROGRAM_SEARCH"
#define WEBAPI_XML_TVPROVIDER_LOOKUP    "TVPROVIDER_LOOKUP"
```

Perform a Text Search

This method performs text searches. Different types of search requests are created depending on which search type the user selects. For example, this can be used to search for a Contributor by Name.

```
void webapi_ref::text_search()
{
    webapi_error_t         error = WEBAPI_OK;
    webapi_object_t        *wobj = NULL;        // The object retrieved from the Gracenote Media Service.
    webapi_query_options_t options;             // The query options.
    QString                xml;
    history_element_t      new_history_elt;

    QApplication::setOverrideCursor(Qt::WaitCursor);

    options.range_start = ui.range_start_spinBox->value();
    options.range_end =   ui.range_end_spinBox->value();

    if (options.range_start < 1)
    {
        options.range_start = 1;
    }

    if (options.range_end < options.range_start)
    {
        // If the range is invalid, set it to something valid.
        options.range_end = options.range_start + 19;
        ui.range_end_spinBox->setValue( options.range_end);
    }

    options.select_extended = generate_select_extended();

    if (ui.checkBox_single_best->isChecked())
    {
        options.mode = WEBAPI_XML_SINGLE_BEST;
    }

    // Search for a cast or crew member by name.
    if (ui.search_cast_radioButton->isChecked())
```



```
    {
        error = webapiclient.search(
            webapi_contributor,
            "NAME",
            ui.search_lineEdit->text(),
            &options,
            &wobj
            );
    }
    // Search for a film or episode by title.
    else if (ui.search_film_radioButton->isChecked())
    {
        error = webapiclient.search(
            webapi_work,
            "TITLE",
            ui.search_lineEdit->text(),
            &options,
            &wobj
            );
    }
    // Search for a series by title.
    else if (ui.search_series_radioButton->isChecked())
    {
        error = webapiclient.search(
            webapi_series,
            "TITLE",
            ui.search_lineEdit->text(),
            &options,
            &wobj
            );
    }
    // Search for a Live TV listing by title.
    else if (ui.search_live_tv_radioButton->isChecked())
    {

        //fill in start and end dates if empty
        if (ui.lineEdit_start_date->text().isEmpty())
        {

ui.lineEdit_start_date->setText(QDateTime::currentDateTimeUtc().addSecs(-3600).toString("yyyy-MM-ddThh:mm"));
}

        if (ui.lineEdit_end_date->text().isEmpty())
        {

ui.lineEdit_end_date->setText(QDateTime::currentDateTimeUtc().addSecs(3600).toString("yyyy-MM-ddThh:mm"));
        }
```

Perform an Object Fetch Using a GN_ID

This method fetches an object, such as an AV_Work, based on its Gracenote ID (GN_ID).

```
void webapi_ref::lookup_id()
```



```
{
    webapi_tv_setup_t      tv_setup;
    webapi_query_options_t options;
    webapi_object_t *      wobj = NULL;
    webapi_error_t         error = WEBAPI_OK;
    QString                id;
    QAbstractButton*       checked_button = ui.lookup_types->checkedButton();
    QString                item_type;
    history_element_t      new_history_elt;

    if (checked_button != NULL)
    {
        item_type = checked_button->text();
    }

    id = ui.id_lineEdit->text().trimmed();

    if (id.isEmpty() || item_type.isEmpty() )
    {
        return;
    }

    QApplication::setOverrideCursor(Qt::WaitCursor);

    options.select_extended = generate_select_extended();

    // Fetch a contributor.
    if (item_type == "CONTRIBUTOR")
    {
    //  tv_setup.enabled_channel_ids.append( ui.tvchannelid_lineEdit->text().trimmed());
        options.tv_setup = &tv_setup;
        error = webapiclient.fetch( webapi_contributor,  id, &options, &wobj);
    }
    // Fetch an AV Work.
    else if (item_type == "AV_WORK")
    {
    //  tv_setup.enabled_channel_ids.append( ui.tvchannelid_lineEdit->text().trimmed());
        options.tv_setup = &tv_setup;
        error = webapiclient.fetch( webapi_work,  id,  &options, &wobj);
    }
    // Fetch a series.
    else if (item_type == "SERIES")
    {
        QTreeWidgetItem*    current_item = NULL;
        QTreeWidgetItem*    mediagraphy = NULL;
        QTreeWidgetItem*    contributor = NULL;
        QTreeWidgetItem*    gnid = NULL;
        QString             gnid_contributor_str;
        bool                do_contributor_series = false;

        // Special case:  What if the current response is a contributor? Offer option to
        // limit the Series results to the current contributor, or show all Series results.
        current_item = ui.result_treeWidget->currentItem();
        if (current_item)
        {
            mediagraphy = current_item->parent();
            if (mediagraphy && (mediagraphy->text(0) == "MEDIAGRAPHY"))
```

```
            {
                contributor = mediagraphy->parent();
                if (contributor && contributor->text(0) == "CONTRIBUTOR")
                {
                    int i;
                    for (i=0;i<contributor->childCount();i++)
                    {
                        if (contributor->child(i)->text(0) == "GN_ID")
                        {
                            gnid_contributor_str = contributor->child(i)->text(2);
                            break;
                        }
                    }
                }
            }
        }

        if (!gnid_contributor_str.isEmpty())
        {
            int result  = QMessageBox::question(    this,
                "Series Fetch",
                "Do you want to limit the Series results to the current contributor, or show all Series
results?\n\n Yes: Show only the Seasons and Episodes of this Series for this contributor\n\nNo:Show the
Seasons and Episodes for the entire Series.",
                QMessageBox::Yes | QMessageBox::No );

            if (result == QMessageBox::Yes)
            {
                do_contributor_series = true;
            }

        }

        if (do_contributor_series)
        {
            options.expand_series = id;
            error = webapiclient.fetch(webapi_contributor, gnid_contributor_str, &options, &wobj);

        }
        else
        {
            error = webapiclient.fetch( webapi_series,  id,  &options, &wobj);  // Fetch the series.
        }

    }
    // Fetch a season.
    else if (item_type == "SEASON")
    {
        error = webapiclient.fetch( webapi_season,  id,  &options, &wobj);
    }
    // Fetch a video disc set.
    else if (item_type == "VIDEODISCSET")
    {
        error = webapiclient.fetch( webapi_product,  id,  &options, &wobj);
    }
    // Fetch a TV program.
    else if (item_type == "TVPROGRAM")
```

```
    {
        error = webapiclient.tv_get_program( NULL, webapi_tvprogram, id, &options, &wobj);
    }
    // Fetch a TV provider channels.
    else if (item_type == "TVPROVIDER")
    {
        tv_setup.enabled_provider = id;
        error = webapiclient.tv_get_channels( &tv_setup, &options, &wobj);
    }
    // Fetch a TV grid
    else if (item_type == "TVGRID")
    {
        int i;
        QStringList channels = id.split(','); //split input on a comma, so we can have multiple channels on
one input line

        for (i=0;i<channels.count();i++)
        {
            QString one_channel = channels[i].trimmed();
            if (!one_channel.isEmpty())
            {
                tv_setup.enabled_channel_ids.append(one_channel);  //put single DVB triplet in the tv_setup
structure
            }
        }

        //fill in start and end dates if empty
        if (ui.lineEdit_start_date->text().isEmpty())
        {

ui.lineEdit_start_date->setText(QDateTime::currentDateTimeUtc().addSecs(-3600).toString("yyyy-MM-ddThh:mm"));
}

        if (ui.lineEdit_end_date->text().isEmpty())
        {

ui.lineEdit_end_date->setText(QDateTime::currentDateTimeUtc().addSecs(3600).toString("yyyy-MM-ddThh:mm"));
        }

        //fill in mode and gn_id for (work/contributor/series/season)-on-tv

        QRadioButton* checked_mode =  dynamic_cast<QRadioButton*>  (ui.tvgrid_modes->checkedButton());
        if (checked_mode && checked_mode != ui.radio_tvgrid_mode_none)
        {
            options.mode = checked_mode->text();
            options.input_gnid = ui.lineEdit_option_gn_id->text().trimmed();
        }

        error = webapiclient.tv_get_grid(&tv_setup, &options, ui.lineEdit_start_date->text(),
ui.lineEdit_end_date->text(), &wobj);

    }


    // Fetch a TV channel batch list..
    else if (item_type == "TVCHANNEL")
    {
```



```
        tv_setup.enabled_channel_ids.append(id);

        //add the stamp (empty stamp is OK)
        tv_setup.enabled_channel_id_stamps.append(ui.lineEdit_channel_stamp->text().trimmed());

        error = webapiclient.tv_get_program_batch_list(&tv_setup, NULL, &wobj);
    }
    // Fetch a TV program batch URL.
    else if (item_type == "TVPROGRAMBATCH URL")
    {
        // Download a TV program batch.
        error = webapiclient.tv_get_program_batch( id,  &wobj);
        new_history_elt.request_xml = "HTTP GET: " + id;
    }
    else
    {
        QMessageBox::information(this, "ERROR", "Cannot look up " + item_type);
        error = WEBAPI_ERROR;
    }

    if (error == WEBAPI_OK)
    {
        // Add Request XML to a new history element.
        wobj->xml_request_string( &new_history_elt.request_xml);



        // Add Response XML to the history element.
        wobj->xml_response_string( &new_history_elt.response_xml);

        // Push the history element into the history list.
        push_new_lookup_history(new_history_elt);

        wobj->xml_object_release();

        render();
    }

    QApplication::restoreOverrideCursor();
}
```

Getting TV Providers

This method looks up the TV providers for the selected postal code for North America or DVB (Digital Video Broadcast) triplet for Europe. In the case of North America, a single postalcode is sent in the query, and the reply will be a set of channels. In the case of Europe, multiple DVB triplets can be sent in a single query, and the response contains a set of channels.

```
void webapi_ref::lookup_zip()
{
    //IPG starts here
    webapi_tv_setup_t      tv_setup;
```

```
    webapi_object_t        *wobj = NULL;
    webapi_query_options_t options;
    webapi_error_t         error = WEBAPI_OK;
    history_element_t      new_history_elt;
    QString                trimmed_input;
    char*                  input = "";

    if (ui.lineEdit_zip_code->text().isEmpty())
    {
        return;
    }

    options.select_extended = generate_select_extended();

    trimmed_input = ui.lineEdit_zip_code->text().trimmed();



    if (trimmed_input.left(3).toLower() == "dvb")
    {

        int i;
        QStringList dvbs = trimmed_input.split(','); //split input on a comma, so we can have multiple dvb
triplets on one input line

        if (dvbs.count() == 0)
        {
            tv_setup.dvbs.append(trimmed_input);  //put single DVB triplet in the tv_setup structure
        }
        else
        {
            for (i=0;i<dvbs.count();i++)
            {
                QString one_triplet = dvbs[i].trimmed();
                if (!one_triplet.isEmpty())
                {
                    tv_setup.dvbs.append(one_triplet);  //put single DVB triplet in the tv_setup structure
                }
            }
        }


        QApplication::setOverrideCursor(Qt::WaitCursor);
        error = webapiclient.tv_get_channels(&tv_setup, &options, &wobj);
        input = "dvb triplet";
    }
    else
    {
        // Use the postal code.
        tv_setup.postalcode = trimmed_input;
        QApplication::setOverrideCursor(Qt::WaitCursor);
        error = webapiclient.tv_get_providers(&tv_setup, &options, &wobj);
        input = "postalcode";
    }


    QApplication::restoreOverrideCursor();
```



```
    if (WEBAPI_OK == error)
    {
        // Add Request XML to a new history elelement.
        wobj->xml_request_string(&new_history_elt.request_xml);

        // Add Response XML to the history element.
        wobj->xml_response_string(&new_history_elt.response_xml);

        // Push the history element into the history list.
        push_new_lookup_history(new_history_elt);

        wobj->xml_object_release();

        render();
    }
    else
    {
        QMessageBox::information( this, "Error", "There was an error retreiving channel information for
selected "  +QString(input) + ".");
    }

}
```

Getting TV Grid Data

The WebApiClient::tv_get_grid method retrieves EPG data to be used for a TV grid displayed to the end user. The reference application shows the grid data as XML. Real application usually show this information in a table (grid) format. This method uses macro definitions to call the Web API query TVGRID_LOOKUP.

```
webapi_error_t WebApiClient::tv_get_grid(webapi_tv_setup_t*  setup, webapi_query_options_t* options,
QString start_date, QString end_date, webapi_object_t** obj)
{

    QDomDocument query_doc;
    QDomElement queries_element;
    QDomElement query;

    int i;

    if ((obj == NULL) || (setup == NULL) || start_date.isEmpty() || end_date.isEmpty())
    {
        return WEBAPI_ERROR;
    }

    // Create QUERIES XML document with AUTH structure
    xml_create_queries_element( &query_doc, &queries_element);


    xml_add_query(&query_doc, &queries_element, &query, WEBAPI_XML_TVGRID_LOOKUP);

    if (setup->enabled_channel_ids.count() == 0)
    {
```

```
        return WEBAPI_ERROR;  //no channels enabled!
    }


    xml_add_channel_ids_to_query( &query_doc, &query, setup, WEBAPI_XML_GN_ID, WEBAPI_XML_TVCHANNEL);


    //add start and end date
    QDomElement start_date_element = query_doc.createElement(WEBAPI_XML_DATE);
    start_date_element.setAttribute( WEBAPI_XML_TYPE, WEBAPI_XML_START);
    start_date_element.appendChild( query_doc.createTextNode( start_date ));
    query.appendChild(start_date_element);

    QDomElement end_date_element = query_doc.createElement(WEBAPI_XML_DATE);
    end_date_element.setAttribute( WEBAPI_XML_TYPE, WEBAPI_XML_END);
    end_date_element.appendChild( query_doc.createTextNode( end_date ));
    query.appendChild(end_date_element);



    if (options)
    {

        if (!options->select_extended.isEmpty())
        {
            xml_add_query_option(&query_doc, &query, WEBAPI_XML_SELECT_EXTENDED, options->select_extended);
        }

        if ((!options->mode.isEmpty()) && (!options->input_gnid.isEmpty()))
        {
            xml_add_mode(&query_doc, &query, options->mode);
            QDomElement gnid = query_doc.createElement(WEBAPI_XML_GN_ID);
            gnid.appendChild(query_doc.createTextNode( options->input_gnid));
            query.appendChild(gnid);

        }

        // Put other options here.
    }

    // Perform network access.
    QString request_str = query_doc.toString();
    QString result = perform_http_post(service_url, request_str);

    // Package up response.
    return xml_package_response(obj, result, (this->save_requests ? request_str : "" ), webapi_tvgrid);
}
```

Looking Up a Video Disc using its TOC

This method looks up an object using its table of contents (TOC). For example, it can look up a video disc or Blu-ray disc by TOC.

```
webapi_error_t WebApiClient::toc_lookup(
    webapi_object_type_t fetch_type,
    QString toc,
    webapi_query_options_t *options,
    webapi_object_t **obj
    )
{
    // Currently, processing options is not implemented.
    QDomDocument query_doc;
    QDomElement  queries_element;
    QDomElement  fetch_query;

    if (obj == NULL)
    {
        return WEBAPI_ERROR;
    }

    // Create a QUERIES xml document with auth structure
    xml_create_queries_element(&query_doc, &queries_element);

    // Add a  *_FETCH query.
    xml_add_toc_lookup(&query_doc, &queries_element, &fetch_query, fetch_type, toc);



    if (options)
    {

        if (!options->mode.isEmpty())
        {
            QDomElement mode = query_doc.createElement(WEBAPI_XML_MODE);
            mode.appendChild(query_doc.createTextNode(options->mode));
            fetch_query.appendChild(mode);
        }


        if (!options->select_extended.isEmpty())
        {
            xml_add_query_option(&query_doc, &fetch_query, WEBAPI_XML_SELECT_EXTENDED,
options->select_extended);
        }

        // Put other options here.
    }

    // Perform network access.
    QString request_str = query_doc.toString();
    QString result = perform_http_post(service_url, request_str);

    // Package up the response.
    return xml_package_response(obj, result, (this->save_requests ? request_str : "" ), fetch_type);
}
```


Examples

This section gives input and output data for query examples, and instructions on testing TV Text Search.

Contributor

Query	Input	Output
Text Search on *Jeff Bridges*	Search Type: Cast and Crew Search String: Jeff Bridges Lookup Options: IMAGE	Short metadata for Jeff Bridges, including Birthday 1949-12-04 and Image URL.

Full Mediagraphy Lookup for *Jeff Bridges*	Fetch radiobutton: CONTRIBUTOR ID
	Lookup Options: IMAGE, FULL_MEDIAGRAPHY, MEDIAGRAPHY_IMAGES
		Full Contributor metadata and mediagraphy, including:
		• Biography
		• Contributor Type: Actor and Executive Producer
		• Image URL
		• Mediagraphy Including:
		  • AV_WORK: Tron Legacy
		  • AV_WORK: True Grit

TV Series Contributor Mediagraphy	Fetch radiobutton: CONTRIBUTOR ID Lookup Options: FULL_MEDIAGRAPHY	Mediagraphy for contributor Robert Patrick. Included should be SERIES: The X Files. At present this is Rank # 61, however it may shift in position if new items are added to his filmography. If you double-click on the X-Files SERIES, the reference applicaiton will ask "Do you want to limit the Series results to the current contributor, or show all Series results?" Click YES. Result:The result will be the contributor's EXPAND_SERIES mediagraphy, which is the list of episodes and seasons of an SERIES that the selected contributor has appeared in. This is in contrast to seeing ALL data on a particular TV Series.

EPG Examples: European

The following DVB triplets resolve to the following channels with EPG data:

Triplet	Channel
dvb://1.1100.8703	13èmeRU
dvb://65024.1100.400	3sat
dvb://1000.3.178	Action
dvb://6.606.2381	Einsfestival

EPG Examples: North American

Query	Input	Output

North America EPG Example: Comcast Sunnyvale	Input Zip Code: 94086 (Sunnyvale, CA) Selected Provider: Comcast Sunnyvale	Example Channel IDS from this provider with EPG data: KTVU Channel 2 KNTV Channel 3 KRON Channel 4
North America EPG Example: San Francisco DISH	input Zip Code: 94553 (Martinez, CA) Selected Provider: DISH San Francisco	Example Channel IDS from this provider with EPG data: Comedy Central Channel 107 Food Network Channel 110

Series and Season

Query	Input	Output
TV Series Text Search: X-Files	Search Type: TV Series Search String: X Files	The output may contain multiple results, but should include this one: The X-Files
TV Series Text Fetch: X-Files	Fetch radiobutton: SERIES ID Lookup Options: IMAGE, CONTRIBUTOR_IMAGE, SEASON_IMAGE	Title: The X-Files DATE: 09-10-1993 Genre: Drama, Mystery, Science Fiction Synopsis Image URL Contributors: (including) Gillian Anderson David Duchovny Seasons: Season 1 through Season 9 Note: Season Image URLs are not currently available, but may be available in the future.
TV Season Fetch	Fetch radiobutton: SEASON ID	Metadata for Season 1 of The X Files. This includes: • SERIES: The X Files • AV_WORK: Pilot • AV_WORK: Deep Throat • AV_WORK: Squeeze

Work

Query	Input	Output
Text Search for *Dark Knight*	Search type: Film or Episode Search String: Dark Knight	Many AV_WORKS. One should have an MPAA rating of PG-13. This is the feature film Dark Knight. The remaining AV_WORKS are TV Episodes with the title Dark Knight.

Fetch *Dark Knight*	Fetch radiobutton: AV_WORK ID Lookup Options: IMAGE, CONTRIBUTOR_IMAGE	Full metadata for *The Dark Knight*, including: • MPAA Rating: PG-13 • Duration: 9120 Seconds • Release Date: 2008-08-13 • Genre: Adventure, Thriller, Action, Drama • Synopsis • Image URL • Contributors: • Christian Bale • Heath Ledger • Michael Caine