SageTV Studio

Version 4.0

User's Guide

Table of Contents

1) Getting Started

Introduction

SageTV's user interface is defined inside an STV file. The interface and file are both created using a special-purpose application called SageTV Studio. The basics of installing and getting started with this application are the focus of this document.

Installation

SageTV Studio is installed as part of a normal SageTV or SageTV Client installation, so no special steps need to be taken to install Studio.

Note: Since Studio operates the same in SageTV or SageTV client, further the use of the term "SageTV" in this document also refers to "SageTV Client", unless otherwise stated.

License Key Requirement

Note that while Studio is part of a SageTV installation, Studio is not available during the trial period. A valid SageTV license key must be entered before Studio may be launched.

Starting Studio

Once SageTV is registered, Studio may be started from within SageTV by using the **Customize** command. The default keystroke for that command is **Ctrl+Shift+F12**, and it may be changed in SageTV by going to: Setup -> Detailed Setup -> Commands. Issuing the Customize command will cause two things to occur:

1. If Studio is not already open, it will be opened in a new window.
2. The Widget that corresponds to the currently showing Menu or OptionsMenu in the SageTV UI will be highlighted in the Studio window.

Additional Documentation

In addition to this user guide, you may wish to consult other documentation regarding customizing SageTV:

- ☮ **SageTV API** – See http://download.sage.tv/api/index.html
- ☮ **Plugin Documentation** – See http://www.sagetv.com/configuration.html

Safe STV Editing

When using Studio, it is recommended that you edit a copy of the STV that you wish to modify so that the original STV remains unchanged and available for reference, in case you make editing mistakes and are not sure how to return the edited STV back to its original condition. **Note:** The default SageTV3.STV is installed as a read-only file, so you may need to turn off that file attribute for any copies you create of that file.

In order to prevent interfering with recordings currently in progress, it is also recommended that you use Studio from a client instance of SageTV. That way, if you have to end the SageTV process (perhaps due to accidentally creating an infinite loop), no recordings will be affected. Editing in a client instance can be done by:

1. Run SageTV in Service Mode. When you do this, the UI portion is run as a client, so exiting the UI instance will not affect recordings that may be in progress.

2. Install SageTV Client and use Studio from an instance of that client. **Remember:** you can run multiple instances of a full SageTV Client installation by using the **-multi** command line parameter when running SageTVClient.exe.

2) The SageTV Studio Language

The STV File in Relation to SageTV

SageTV consists of two parts: a **core program** that defines how SageTV runs and what its basic capabilities are, and an **STV file** that defines the User Interface (UI).

The **core program** does not directly interact with a user; it provides all the "behind the scenes" functionality such as controlling capture devices, creating the recording schedule, recording the scheduled shows, deleting recordings to make room for new ones, and many other responsibilities. In order for the core program to display an interface that a user can interact with, it needs an STV file, which defines the entire UI. Without an STV file, the user will see a blank screen in the SageTV window or display area and will have no way to make use of SageTV's features. To make the capabilities of SageTV available to an STV file, there is a set of API functions that the STV code can call to access the core's data and functionality.

The **STV file** uses basic screen display elements and the core's API calls to create and display the entire UI that a user interacts with. When using SageTV, every visible UI feature is defined in the STV that is in use. Any of the options that the core can support must be presented in the STV in order for a user to have access to those options. So, an STV can provide access to as much of SageTV's core capabilities as a developer wishes to make use of.

Studio is the application used to develop and maintain these STV files.

The Widget

What is a Widget?

An STV consists of **widgets**, which are the basic building blocks of the Studio language. There are 18 types of widgets that provide UI display and code execution functionality. These widgets are not like lines of code in the traditional sense of a programming language such as C++ or java; rather, they define what the SageTV core will do when a widget is encountered/executed: the type of the widget defines what SageTV could do at that point, while the details of the widget (its properties; more on that later) tell SageTV exactly how that widget is to be implemented.

Widget Relationships

While a text-based programming language has lines of code that follow one after another and are executed in that order, widgets have parent ↔ child relationships, or **references**.

A series of widgets is executed from parent → child → next child, and so on, so that execution flows from a parent widget down through all of its children. A parent widget can have multiple children, and each of those children may have one or more children. There is a defined order to children, so child 1 comes before child 2. This sequence will be referred to as a **widget chain**.

Each child widget is also capable of having multiple parents. There is no execution order regarding the parental connections. The consequences of having multiple parents will be discussed later.
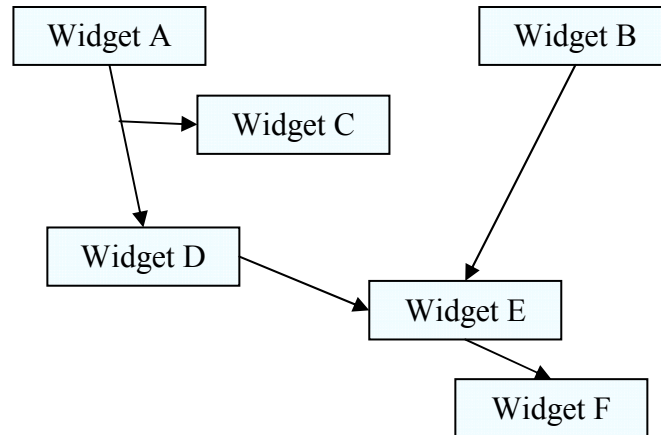
Consider a set of abstract widgets: A, B, C, D, E, and F. For this example, it doesn't matter what kind of widgets these are; just consider them to be generic widgets, Studio's basic building block.

Widget Chain for A

Widget A is a parent of widget C. Widget A is also a parent of widget D, which is a parent of widget E, which happens to be the parent of widget F. Note that A has two children, C and D, and that child C is A's first child, since it branches off first. In this case, the widget order is A → C → D → E → F.

Widget Chain for B

Widget B is a parent of widget E, which is a parent of widget F. For this widget chain, the widget order is B → E → F.

Note that E has two parents: D and B. There is no first or second parent in terms of execution priority, since execution flows from parent to child; E simply has two parents. (However, as explained in the next section, there is a distinction regarding the primary parent only for display purposes in Studio.)

What Studio does is to take these abstract internal widget relationships and display them in a visual manner that shows hierarchy and relationship in an understandable visual

manner: an expandable tree, which branches from parent to children as sections of the tree are expanded.

Displaying Widget Relationships as a Tree

In Studio, the widget relationships are visually displayed as an expandable tree, similar to how a directory tree structure is displayed in an application such as Windows Explorer. When a parent widget is expanded, its child widgets are displayed indented and below the parent in the order of their relationship to the parent: the first child is shown first, second child shown second, etc. If any of those child widgets have children, the display can be expanded to show them indented below the child also. In this way, the widgets appear to have an order, similar to a text programming language having its lines shown in order in a text editor.

When a widget has multiple parents, one of that widget's occurrences will be chosen as the **Primary Reference** and its display will be **bolded** in Studio. Every other reference to that widget will be shown in *italics*. The **bolded** widget may be expanded to see its children in a tree display, as described above; the *italicized* widget cannot be expanded, even though it has the same exact children. (One reason that its children cannot be displayed is that code loops are possible via references, and an expanded loop could run on indefinitely.) There is no other difference: all connections to child widgets are done via references internally to SageTV; it is just that multiple references are displayed in this way to distinguish the fact that there are multiple references to that widget. **Tip:** To see the children for an *italicized* widget, double click on it with the mouse to jump to its primary reference.

Now let's consider the series of abstract widgets mentioned previously:
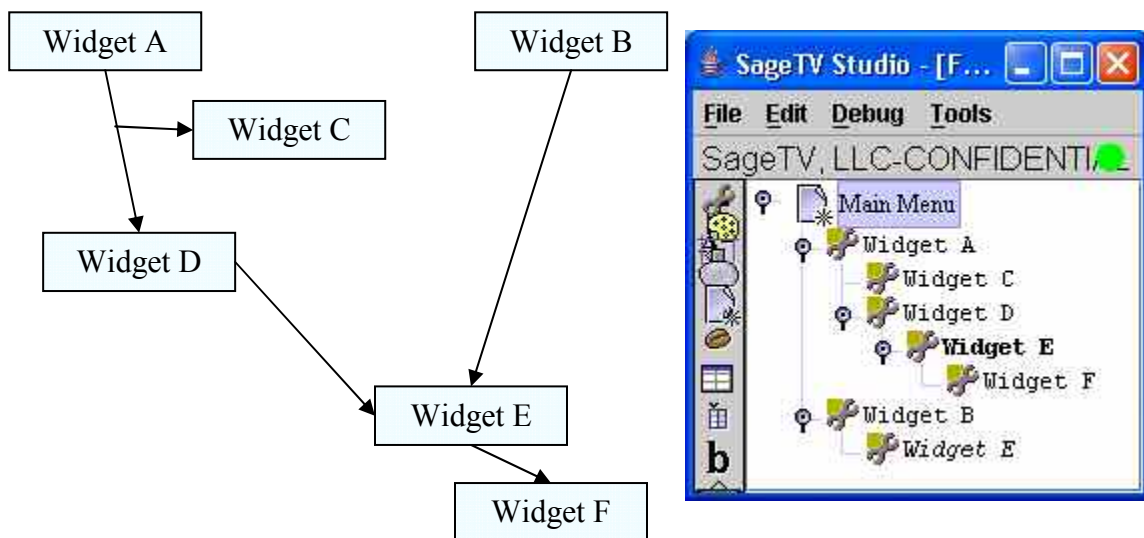
The same series of generic widgets is now shown as they would appear in Studio.

Widget Chain for A

In the Studio image, notice that the widget chain for parent A shows both the order (A → C → D → E → F) via the display order, and the parent ↔ child relationships via the expanded & indented tree display format.

Widget E is **bolded** below widget D, because that reference was chosen as the primary reference. The primary reference is where all of that widget's children are shown, so widget F can be seen directly below E.

Widget Chain for B

Again, notice that the widget chain for parent B shows both the order (B → E → F) via the display order, and the parent ↔ child relationships via the expanded & indented tree display format.

Widget E is *italicized* below widget B, because that reference is not the primary reference. Note that the entire tree has been completely expanded and that widget F is only displayed as a child of widget E below E's primary reference. It is still a child of E for widget B's child widget chain, even though it isn't displayed there – remember: it is connected via the *italicized* link. The **bolded** and *italicized* versions of widget E are actually the exact same widget; it is just that for display purposes, one has been chosen to be the one where its children can be shown.

Note: The Top level of an STV, as displayed in Studio, is a special situation: the top level does not indicate code execution order; it simply lists the root items in alphabetical order for ease of finding the menus, hooks, etc., that are listed there. In the example above, only the **Main Menu** widget is at the top level.

Widget Types

The different types of widgets are more thoroughly discussed later in this document (see 3) Widget Details), but to introduce the terms, the basic widget types available are:

- **Menu** – The top-level UI Widget. (see Menu Widget)
- **OptionsMenu –** Provides 'pop-up' menus for SageTV. (see OptionsMenu Widget)
- **Panel** – A rectangular UI element container. (see Panel Widget)
- **Theme** – Used to apply a general appearance to a UI Widget hierarchy. (see Theme Widget)
- **Action** – Actions are expressions that are executed as SageTV runs. (see Action Widget)
- **Conditional** – Used to conditionally execute an action chain or conditionally show a UI component. (see Conditional Widget)
- **Branch** – Used for multiple-option **Conditional** branching – similar to switch/case in C/C++. (see Branch Widget)

- ☻ **Listener** – Used to respond when the user issues a command. (see Listener Widget)
- ☻ **Item** – A UI element that can have 'Focus' and can be selected by the user. (see Item Widget)
- ☻ **Table** – Used to create dynamic, scrollable UI components. (see Table Widget)
- ☻ **TableComponent** – Used inside a Table to specify the elements of the **Table**. (see TableComponent Widget)
- ☻ **Text** – Used to display text. (see Text Widget)
- ☻ **Image** – Used to display an image. (see Image Widget)
- ☻ **TextInput** – Used for text entry. (see TextInput Widget)
- ☻ **Video** – Used to display video. (see Video Widget)
- ☻ **Shape** – Used to draw basic geometric shapes. (see Shape Widget)
- ☻ **Attribute** – Declares a variable to store data for later reference. (see Attribute Widget)
- ☻ **Hook** – Used like a callback system where the UI can respond to certain events that happen in the core. (see Hook Widget)

Widget Chain Types

As mentioned previously, each parent widget is capable of having multiple child widgets. The multiple child widgets have a defined order in their relationship to the parent: there is a first child, second child, third child, etc. A series of widgets from parent → child → next child, etc., is a **widget chain**, and widget chains are executed in the order of parent, then first child and all of its children, then the next child and all of its children, and so on, until all of the parent's child widgets have been processed. Thus, in programming terms: execution follows depth then breadth.

Widget chains can be one of three types, depending on the purpose of the widgets in the chain. **Tip:** certain types of widgets, such as Actions, Conditionals, and Branches, have a color coded indicator displayed to the left of the widget in Studio.

- ☻ **Process** (green indicator) – Process action chains consist of widgets used only for code execution purposes; they do not lead directly to widgets that display a UI element, except for Shapes, Menus, and OptionsMenus. A Process action chain may be executed as a result of a hook, listener or a user selection of a UI element.

- ☻ **UI** (blue indicator) – UI action chains consist of a series of action widgets that lead directly to a child UI element widget. If an widget chain in this situation does not have a UI widget in its child tree, and is not in a process chain that gets executed as a result of user interaction (Process chain), then that action will not get executed and will not be marked with a blue indicator. **Note: if there is a line of code not being executed as part of the data manipulation when creating a display, check for any widgets marked yellow (see next action chain type.)**

- ☻ **None** (yellow indicator) – If a widget chain is not part of a Process or UI chain, then it widgets will not be executed. This usually happens in a UI action chain,

where some action widget is not a parent of a child chain of widgets containing a UI element. As mentioned above: in a UI widget chain, SageTV will only execute those actions that are in the UI element's sequence of parent widgets.

Advanced Note: In certain situations, it is possible to have widget chains that are part of both a Process and UI chain. Example: If there is UI chain code that is referenced elsewhere as part of a Process chain, some of the UI chain's code can still be executed as part of that referenced code. However, any code under a UI element widget will not get executed as part of the Process chain in that referenced code. This is used in the default STV in order to allow the use of the same code to display the System Information on the screen when the menu is drawn, and to have the same data creation widgets used when writing the system configuration information to a data file as a result of the Info command listener. (See the System Information menu in the standard V4 STV.) **Note: In such shared code, a Process chain will not continue executing code below a UI element widget (such as a Panel) that is in its tree. Any widgets below the UI element will only be part of the UI chain.**

Expressions

General Expression Information

Code that is displayed on widgets such as Actions, Conditionals, and Branches are expressions that SageTV will evaluate (execute) at run time. Such expressions may be part of either Process or UI action chains. To easily see which widgets will be evaluated, simply look at how a widget's name is displayed in Studio: if it is displayed using a fixed width font, then it is an expression that will be evaluated. **Note:** In addition to these widget names, most widget properties that have an entry field can use an expression to dynamically control that property. See General Widget Properties.

Expressions can be simple math or string manipulation or may contain more complex code that calls one or more SageTV API or java functions (see next section). The expressions follow the standard rules of operator precedence.

SageTV will automatically convert most variable types as needed, if possible. Conversions between primitives and Strings will be done as needed. Strings and File objects will also be converted between as needed based on the file path.

Certain SageTV variable types will automatically convert also. Example: Since the MediaFile variable type has a 1-to-1 correspondence to the Airing variable type, a MediaFile can only be associated to a single Airing, and vice-versa. Therefore, SageTV will be able to automatically convert between these two types as needed. A Show, however, could have several Airings, while an Airing always refers to a single Show. So, an API call that expects an Airing as a parameter would not be able to accept a Show variable in its place, since there is no way to guarantee that a Show refers to one specific

Airing. More details on automatic conversion of SageTV object types can be found in the API documentation for the different object types.

Variable Assignment

The general format for variable assignment on an Action widget is:

{<variable name=}<expression>

A variable name followed by an equal sign is optional, but if it exists, then the results of the expression will be stored in the named variable. Regardless of whether an assignment is part of the statement, the results of the expression will be placed in the '**this**' variable, which can be accessed only by the very next expression. **Note:** there can be only one assignment (equal sign) in a statement.

For more details about variable usage, see 4) Attributes / Variables.

Creating Code Comments

Currently, there is no "Remark" type widget. An easy way to add a comment line to STV code is to add an Action widget and place your comment inside a pair of double quotes. To help clarify that the Action is a comment, rather than functional code that is creating a string for use by the Action's child widgets, it is useful to add some sort of string to the beginning of the comment to indicate its purpose; "REM" works well for that purpose. Example: "REM This is a comment"

3) Widget Details

An STV file consists of a hierarchy of Widgets. The various types of widgets and their properties are described below.

General Widget Properties

Each widget has properties that may be set. In many cases, a property value can be **dynamic**: instead of simply listing a value, use an expression instead, such as "=SomeVariable" or "=If(SomeExpression, value-if-true, value-if-false)".

Most of the sliders will adjust a property value from 0.0 to 1.0. Those **decimal** (float) values indicate a percent amount relative to the size of space allocated to that UI element, where 1.0 is 100%. Sometimes, negative or positive decimal values may be entered to extend an UI element beyond its allocated space, but its display will be cut off at its border. Example usage: to create a partial box with a rounded bottom and square top, set its height greater than 1.0 and extend its top above where it may be displayed. The rounded top corners will not be visible.

When a property is given a **integer** value instead of a decimal, it is taken as an absolute value: specifying a height of "1" means 1 pixel; a value of "1.0" means 100% of the display space allocated for that element.

When values are left blank, SageTV will use its default value for that property, automatically calculating locations and sizes. The **preferred size** of a component is the size defined by the properties; or if the sizing properties are not defined then SageTV will calculate a preferential size for display (i.e. for a Text Widget it'll be the size of the text).

Properties Common to Many Widgets

Ignore Theme Properties – Check this option to have a widget ignore the properties defined for that widget type in a Theme and to use its own properties instead. (This will be explained more in the Theme Widget section later).

AutoArrange – This setting affects the layout of any child UI elements. If nothing is selected, the child elements must all specify their own locations. **SquareGrid** will cause the child UI elements to be placed in a square layout grid, with all elements given the same size. **Horizontal** will place all child UI elements in a horizontal row, but the elements will not all be given the same size; they will be sized according to their preferred size. Similarly, the **Vertical** layout will place all items in a vertical column, and size each of them according to their preferred size. A **HorizontalGrid** or **VerticalGrid** layout will place all child UI elements in a horizontal row or vertical column, with all

elements given the same size based on the number of children – the available space will be evenly divided between all children.

Anchor X, Anchor Y – For a widget that can control its own placement, these values determines the horizontal and vertical locations within its display area where the UI element will be placed. At times, a parent widget will automatically control the placement of its children, so this value will be ignored in those conditions. Example: a panel could AutoArrange its children in a HorizontalGrid, in which case, that grid layout will determine where the children are placed.

Fixed Width, Fixed Height – Use these properties to set the width and height that a widget will occupy, relative to the display space allotted by its parent.

Anchor Point X, Anchor Point Y – These properties control the point within the widget that will be placed at **Anchor X** and **Anchor Y**, if values have been provided for the anchor locations. By default, there is no anchor point and the Anchor values are relative to the overall component.

Pad X, **Pad Y** – These values control the spacing between child components in the X/Y dimension; it only applies if they are automatically laid out using Horizontal, Vertical, etc., for AutoArrange. Floats are relative to the size of the entire UI, instead of the parent (position/sizes are relative to the parent). Integers are absolute.

Insets – This is the border inside a component for where the children can be placed. Floats are relative to the size of the entire UI, instead of the parent (position/sizes are relative to the parent). Integers are absolute. If it is one value, it applies to all four sides; if it is 4 values (ex: 0.1,0.2,0.1,0.2) then it applies to top, left, bottom, right respectively.

Vertical Alignment, Horizontal Alignment – These values determine where children will be placed inside this widget's UI area when that area is larger than the size of the children. When using Horizontal arrangement, use **Vertical Alignment** to control the vertical placement of the children when there is extra vertical space. Similarly, when using Vertical arrangement, use **Horizontal Alignment** to control the horizontal placement of the children when there is extra horizontal space.

Animation – This property allows a widget to automatically update the display of itself and all of its children. There are two ways to specify an animation property:

1. The first is using a Start,Period,Duration value for the property. This is just 3 non-negative integers separated by commas. The **first value** indicates the delay after initial display of the menu to begin animation. The **second value** is the time between animation updates. And the **third value** is the time the animation should last for after it has begun; 0 implies forever. All 3 values are in milliseconds. An example is 0,1000,0 which means animate that component every second.

2. The second way to do Animation properties is with dynamic properties (dynamic properties are indicated by starting with the equals sign). With dynamic properties, animation will occur whenever the expression after the equal sign evaluates to true. (More will be explained on this later.)

RenderXform(ScaleX), RenderXform(ScaleY) – These are percent values, from 0.0 – 1.0, which scale the entire widget and its children, keeping all relative positions the same. Ex: setting both values to 0.5 will scale the widget and its children to 50% of full size.

Properties Dialog Buttons

While the properties dialog is open, make any desired changes, then select **OK** to accept the changes and close the properties dialog.

Select **Apply** to apply the current changes.

Select **Cancel** to close the properties dialog, discarding any changes that have not been applied.

To discard all changes since the properties dialog was opened, select **Revert**; if any previous changes had been applied, select **Apply** after reverting.

Menu Widget

Menus are top level widgets and may only occur as top level items; all other occurrences are references to menus and function as menu transitions.

Each screen in the SageTV UI is based on a single Menu widget and only a single Menu widget may be active at any time. All child widgets for a Menu widget are part of that menu.

References to Menu widgets in Process Chains cause a menu transition to occur from the current menu to the one referenced. During that menu transition, all elements of the current menu are cleared, including any active Options Menus and local variables, before loading the new menu. Local variables may be transferred to the next menu by first calling **AddStaticContext** before the menu transition.

Note: Menu contents are cached and reused when the **Back** and **Forward** commands are used. In addition, the last few states of a menu are cached for a few sets of the static context settings used when the menu is called, enabling a faster loading of the menu the next time it is referenced. This means cached Menus will 'remember' where you were on them last the next time they are loaded.

Special Menu Widget Names

Main Menu – The Main Menu is the initial menu that SageTV loads at startup or when awoken from sleeping.

Screen Saver – This Menu is the one loaded when the SageTV screen saver is activated.

Server Connection Lost – When the client loses the connection to the SageTV server, it loads and displays this menu.

Menu Widget Properties

The properties dialog for a Menu is fairly simple and consists of the widget's name and a check box to enable a Video Background for the menu.

The name can be any descriptive text, keeping in mind the special function menu names listed above.

If Video Background is checked, then the menu will show full screen video as the menu's background. This full screen video will completely fill the SageTV window and will ignore any UI overscan setting. (The Detailed Setup -> Multimedia -> Overscan Settings can be used to resize the UI so that it fits on a TV screen, while the video background will ignore that setting and continue to fill the entire size of the SageTV window.)

OptionsMenu Widget

An OptionsMenu is used to pop up a dialog that is active on top of the current Menu. Multiple OptionsMenus may be layered on top of each other. A call to the CloseOptionsMenu() function will close the most recently created OptionsMenu; if there are multiple OptionsMenus active, then each will require a call to CloseOptionsMenu() in order to close them all. Alternatively, all active OptionsMenus will be cleared when transitioning to another Menu.

When an OptionsMenu is closed, the Widget Action chain that caused the OptionsMenu to open will continue execution at the point after where the OptionsMenu was created. This allows OptionsMenus to be used to interactively ask the user questions while processing an Action chain.

Note: When an OptionsMenu ends, continuing execution will not recurse further up the tree than the OptionMenu's direct parent, so execution will only resume for actions that are children of the OptionMenu's direct parent. This is currently logged as a bug in SageTV and will be fixed in a future release. This issue should have minimal impact on the user.

OptionsMenu Widget Properties

Note: OptionsMenu sizes are relative to the entire Menu, not the OptionsMenu's parent widget. The sizes of other widgets are relative to the size of their parent UI elements.

The properties dialog for an OptionsMenu is shown to the right. The first entry is the **name** of the widget. This name is only useful for keeping track of the various OptionsMenus that are used; the names have no special meaning for SageTV.

The remaining properties were already covered above; see Properties Common to Many Widgets.

Panel Widget

A Panel is used as an organizational component for UI elements. It is displayed within a portion of its parent's display area and all of its child UI elements are contained within its own borders. Panels are only used in UI portions of the STV tree; they are never part of the Process chain.

Panel Widget Properties

The name of the panel is unimportant to SageTV; it is purely for use by the Studio developer to keep track of the various panels in use.

Most of the Panel's properties were already covered above; see Properties Common to Many Widgets, but there are a few additional properties:

Scrolling – Determines whether the panel can be scrolled to see additional UI elements that don't fit within the panel's allotted space. If no scrolling type is selected, then the panel's contents may not be scrolled. Choose **Vertical** or **Horizontal** to allow scrolling the panel in one of those directions. Choose **Both** to allow scrolling in both directions.
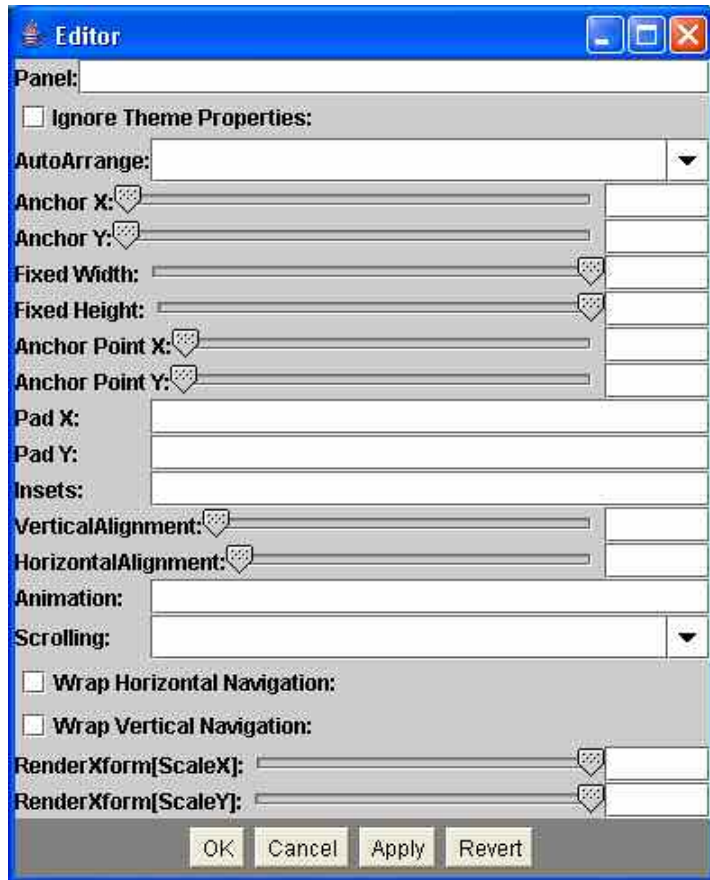
Wrap Horizontal Navigation, Wrap Vertical Navigation – These choices determine which items SageTV will change focus to when an item at an edge of the panel is highlighted and the user uses the directional arrow keys to navigate to another on-screen item.

If these options are checked, then SageTV will wrap navigation back to the other side of the panel and highlight the next item from that direction. Focus stays within the current panel. Ex: Let's say there is a series of buttons in the panel shown in a vertical column, with the top button highlighted. If the user presses the Up arrow, navigation will wrap to the bottom button in the panel and highlight it. If there happened to be another panel

above the current one and it also had buttons, navigation would still wrap back to the bottom button of the current panel rather than going to a button in the upper panel.

If these options are not checked, then SageTV will not wrap navigation only in the current panel. Focus can switch to an item in another panel. In the above example, if there is a button in another panel above the top button in the current panel, SageTV would navigate to it when the Up arrow is pressed.

Theme Widget

A Theme is used to define default UI properties for other widgets; instead of having to define the properties for every single widget, themes may be used to automatically apply properties to any widget which that theme applies to. In addition to the properties of a theme widget itself, themes can have child widgets. Those child widgets become themes for those widget types, so that the properties applied to them are automatically used for any widget of that type within the scope of the theme.

Theme scope: When a UI element widget looks for a theme, it searches for a theme as its immediate child, then searches up through its parent tree until it reaches the Menu widget it belongs to. The first themed widget of its type found in that bottom to top search is used as the theme for its properties. If no theme for its widget type is found it will simply use default values for its properties. **Note:** Panel themes do not recurse in this way; panels only get their theme properties from the theme that is its immediate child.

Remember: Widgets where the Ignore Theme Properties property option is checked do not get their property values from any theme.

Special Theme Widget Names

Global – The theme named "**Global**" is the global theme that applies to the entire STV. It can be placed anywhere in the STV tree. If there is more than one theme named "Global", then the first one found will be used as the global theme. (The "first one found" may not be the one you expect, so it is best to only have one.)

Theme Widget Properties
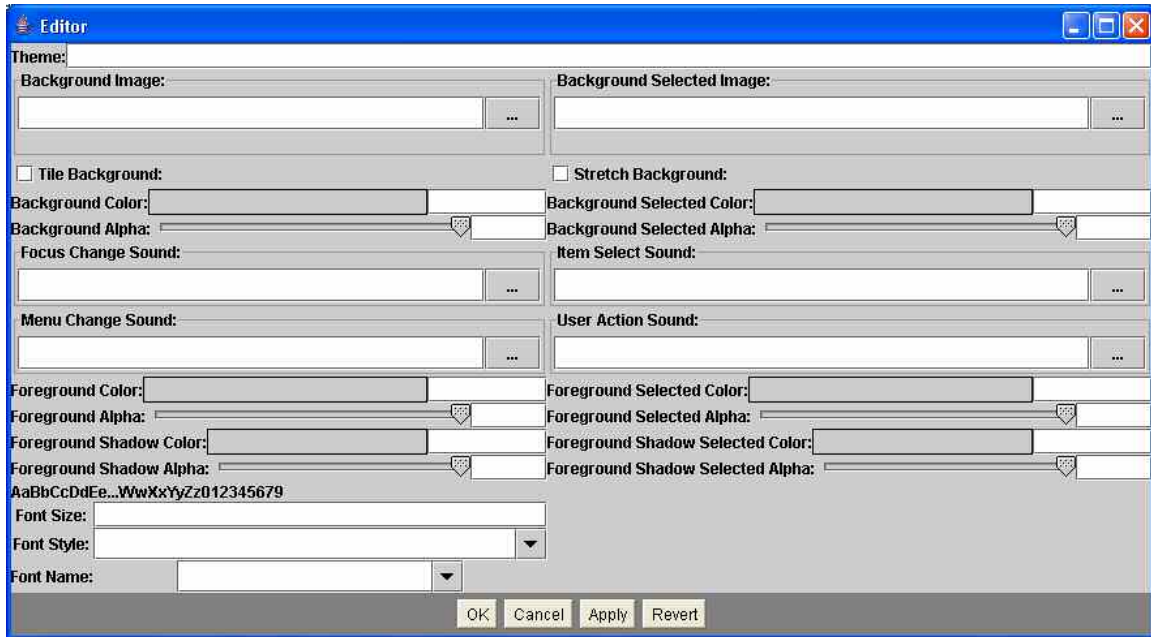
The Theme can be given any name, with only the name "Global" acting as a special global theme, as previously mentioned.

Background Image, Background Selected Image
These entries define the background images only for the widget that is the theme widget's direct parent. The 'selected' image is displayed only if the widget is selectable and has focus.

Tile Background, Stretch Background
Choose whether the background image(s) should be tiled or stretched to cover the entire widget's area. If both items are checked, then the background will be stretched; if neither option is checked, then the image will be simple displayed as-is in the center of the widget's display area.

Background Color, Background Alpha
Background Selected Color, Background Selected Alpha
These are the normal and selected colors used as the background fill for an entire widget's area and the alpha-blend value, from 0 – 255. At alpha 0, the background color is fully transparent; at 255 it is fully opaque. If no alpha is specified, the default setting is fully opaque (255). As with the background images, the background colors only apply to the theme's direct parent widget. Select the color preview box to open a color selection dialog. You may also use a hex value or a dynamic expression to specify the color.

Note: It is best to have the Menu's theme have a fully opaque background – either an image that covers the entire SageTV Window, or a color that fills the area with an alpha

setting of 255. If the Menu has a transparent background, or one that does not cover the entire display area, portions of the previous display may remain visible when the menu gets redrawn.

Property Inheritance Note: The above background-related properties are not inherited for themes further down in the widget chain, but the properties listed below are inherited by other theme widgets which are in the widget tree affected by a theme widget specifying those values. Thus, if any themes in the child tree do not specify the values below, they will use any values specified by the next applicable theme higher in the tree.

Focus Change Sound, Item Select Sound, Menu Change Sound, User Action Sound
These are the sounds to be played when the described events occur. Note that the sounds only function when they are in a theme that is the direct child of a Menu widget.

Foreground Color, Foreground Alpha
Foreground Selected Color, Foreground Selected Alpha
These properties define the normal and selected colors for text and their alpha-blend values, from 0 – 255. Select the color preview box to open a color selection dialog. You may also use a hex value or a dynamic expression to specify the color.

Foreground Shadow Color, Foreground Shadow Alpha
Foreground Shadow Selected Color, Foreground Shadow Selected Alpha
These properties define the normal and selected colors for text shadows and their alpha-blend values, from 0 – 255. Select the color preview box to open a color selection dialog. You may also use a hex value or a dynamic expression to specify the color.

Font Size, Font Style, Font Name
These three properties specify the font that is to be used for any text the theme applies to. For the Size, you may specify a numeric value or use an expression. For the Style and Name, either select an entry in the drop-down list or use an expression that evaluates to one of those entries.

Action Widget

Action widgets contain executable code and may be part of a Process or UI chain. Actions may contain expressions, as discussed in a previous section of this document.

Note: To create a simple comment line, place your comment on an action widget, inside quotes, such as: "REM This is a comment". Just remember that this string will still be evaluated so it is not a true comment since it does affect the code.

Action Widget Properties
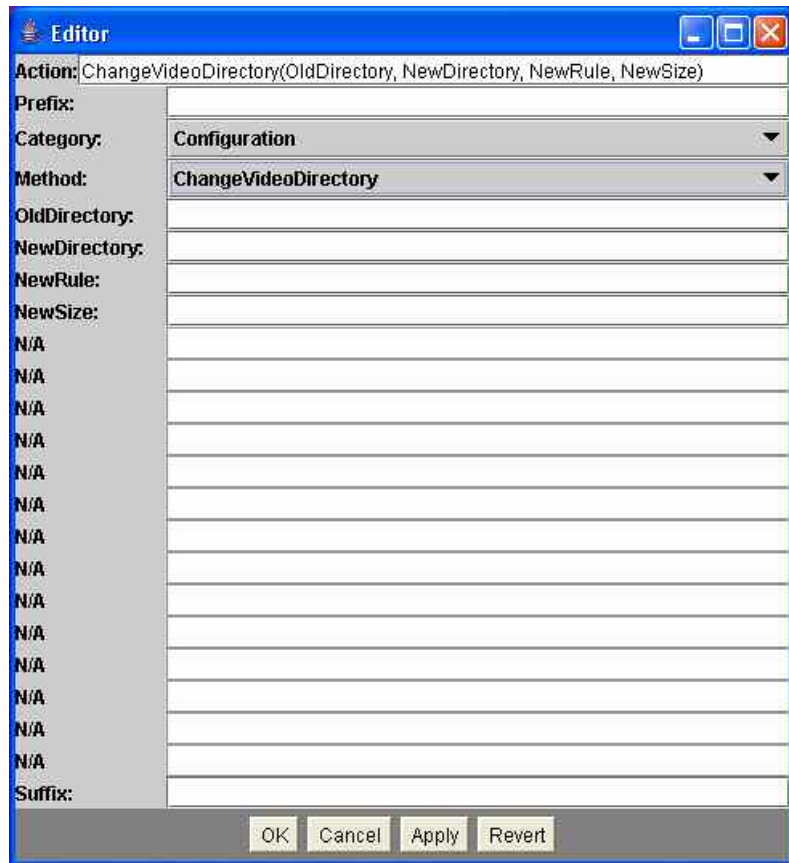
The properties dialog for an Action widget displays the entire text of the action as its 'name', then attempts to parse the expression for the SageTV API it may contain. (Consult the SageTV API documentation for full details on the SageTV API; See http://download.sage.tv/api/index.html) When such an API call is found, it lists all text before the API as the **Prefix**, shows the function's **Category**, displays the function name as the **Method**, then lists all the parameters for that function, followed by all text that is found after the function, placed in the **Suffix** property.

If you wish to simply type in the entire expression yourself, simply highlight the action widget, press F2 to enter edit mode, and type your desired expression. However, the API parsing in this properties dialog provides some assistance in making sure all parameters have been entered and the function name is correct.

Conditional Widget

Conditional widgets are displayed as "**if**" lines in the STV code tree. They function similar to Action widgets, in the sense that they contain executable code and may be part of a Process or UI chain.

The expression contained in a Conditional widget is evaluated, then execution continues depending on the results of that expression and on the type(s) of the Conditional's child widgets.

If the Conditional has Branch Widgets as children, then the result of the Conditional's expression is compared to the results of all of the Branches' expressions. All Branches whose results match will have their children executed. i.e.: multiple branches are allowed to be executed as a result of this comparison. If none of the Branches' expression results match, then the Branch named "else", if there is one, will be executed.

In this example using Branch widgets, "Next Expression 1" and/or "Next Expression 2" could be executed, depending on whether the results of "BranchExpression1" and/or "BranchExpression2" match the results of the Conditional's "Expression". The "else" branch's children will only execute if neither of the other two branches' results match.
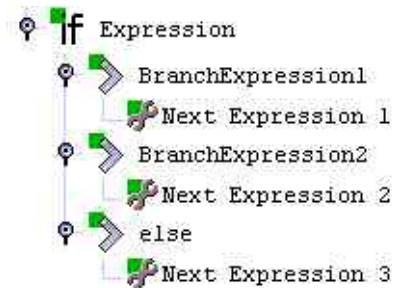
Note: this means that all of the Branches' expressions are evaluated, so don't plan on the second one not being evaluated if the first one's result matches the Conditional.

If the Conditional has no Branch widget children, then its children will be executed only if the Conditional's expression evaluates to "true". Thus:

← **is the same as** →

Conditional Widget Properties

The properties dialog for a Conditional widget is the same as that for an Action widget.

Branch Widget

Branch widgets are discussed along with the Conditional widgets, above. Essentially, Branches control where execution continues after a Conditional widget is evaluated. See above for more details.

Special Branch Widget Names

A branch that is titled **else** will pass code execution to its children if no other Branch expression results match the Conditional expression result.

Branch Widget Properties

The properties dialog for a Branch widget simply consists of the widget's name, which is an expression to be evaluated.

Listener Widget

Listener widgets are activated by user (or command) input, either SageTV commands or one of the special case events such as mouse, raw keyboard, or raw infrared input.

When an event occurs, SageTV checks for UI components that can receive focus, then sends the event to the focused component. The tree hierarchy is searched for a listener for the type of event, starting at the focused component and continuing up the tree through its parents. If that listener type is found, its child widgets are executed. If no listener is found, then the default functionality for that command, if any, is executed in the core program instead. Remember: some commands have default functionality, but not all do.

Once a listener is done executing its code, processing for that event stops, unless the listener includes an action that calls PassiveListen(). When PassiveListen() is called, the event continues processing up the tree hierarchy; successive Listeners for that event could continue passing along the event through the PassiveListen() function, until the event's default functionality is processed in the core.

Listeners that are to be active for the entire STV may be placed in the **Global** theme.

Dual-Use Command Listeners

A dual-use command is one where two commands are listed as part of a single command name, such as "Right/Volume Up". When SageTV receives a dual-use command, it checks to see which command listener is available. For such a command, at a single level in the STV, SageTV checks for the existence of the following listeners, in this order:

1. If the dual-use command's listener is found, that listener is used. In this example, the listener would be for "Right/Volume Up".
2. If the dual-use command's listener is not found, then SageTV looks for a listener for the command listed first. For this example, that listener would be for the "Right" command.
3. If the first command's listener is not found, then SageTV looks for a listener for the command listed second. For this example, that listener would be for the "Volume Up" command.
4. If none of those three listeners are found, then SageTV continues searching up the tree hierarchy or checks for default functionality in the core.

Note that the above applies only when issuing a dual-use command. If a single command is issued instead, the dual-use command's listener is not used. Thus, if the "Right" command is received, SageTV will only check for the "Right" listener; it will not use the "Right/Volume Up" listener.

Mouse Event Listeners

Because a mouse pointer may be moved anywhere on the screen, Mouse events go to the listener for any UI element, not just those that may have focus; i.e.: the mouse can click on non-selectable UI elements and those elements' listeners may respond to mouse input. An example of this in the default STV is the volume bar in the media player, where the volume can be set directly with a mouse click on the volume bar.

Listener Widget Properties

The properties dialog for a Listener widget consists of the widget's name and the Listener Type.

If the Listener's name is blank, then the Type becomes the displayed widget title.

If the Type is blank, then the Type list is searched for one that matches the given Name. If no match is found, then the Listener will not be active.

For the non-command listeners (MouseClick, MouseDrag, RawKeyboard, RawInfrared, Numbers), the Name becomes the local variable for the information associated with the event. The value is also stored in the '**this**' variable. Example: A Listener named **Number** with the Type **Numbers** will have the entered number stored in a local variable called **Number**. For more details, see Listeners That Set Local Variables.

Item Widget

Item widgets are similar to Panels, except that Items can have focus, so they can be highlighted when moving focus via the arrow keys, and they may be clicked on or selected. When an Item is selected, its child action Process chain is executed.

In general, the Item widgets are presented to the user with the appearance of 'buttons' in the default STV (accomplished through the use of themes that apply to the Items).

Item Widget Properties

The properties dialog for an Item is shown to the right.

The first entry is the **name** of the widget. If the Item has no UI widget children, then as a convenience to the developer, the name of the Item will be displayed on the screen as the text for that UI element.

The remaining properties were already covered above; see Properties Common to Many Widgets.
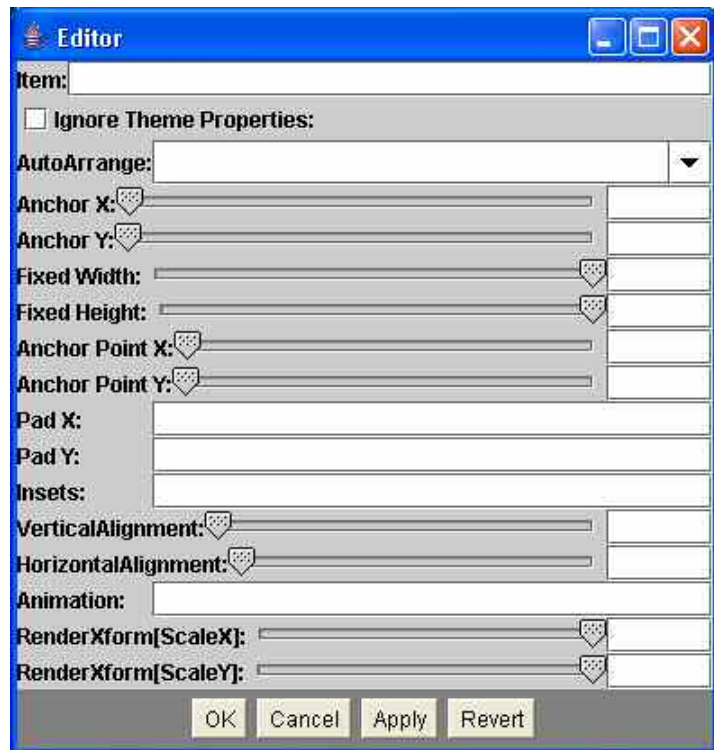
Table Widget

Table widgets are containers for TableComponent widgets, allowing the creation of scrollable UI elements based off dynamic data, such as a list of items from the database that a user can select. An example would be the list of recordings on the SageTV Recordings menu. Since a Table is used in conjunction with TableComponents, details of the combined use of these widgets is covered in the next section: TableComponent Widget.

Table Widget Properties

The properties dialog for a Table is shown to the right.

The first entry is the **name** of the widget. It is used solely as a way for the developer to track Tables in the STV code.

Many of the remaining properties were already covered previously; see Properties Common to Many Widgets. However, there are some properties that are specific to Table widgets.

NumRows and **NumCols** are used to specify how many rows and/or columns are displayed within the widget's portion of the screen. If a value is 0, but the table is set to use that dimension, then the number of elements in the table will be used for that dimension. The space allotted to each row or column is the total space available in that dimension divided by the number of elements to display in that dimension. i.e.: the space is spread evenly so that each item is the same size.

Dimensions is used to specify the direction the table will scroll: **Vertical**, **Horizontal**, or **Both**. A **Vertical** or **Horizontal** table is a 1-dimensional table in the specified direction. If **Both** is selected, the table will be 2-dimensional. The list of SageTV Recordings is a 1-dimensional vertical table, while the LiveTV Guide is a 2-dimensional table.

Wrapping is used to determine whether the table elements will wrap to the beginning or end after scrolling to last or first item. If nothing is selected, then the table will not wrap between the first and last items; once you reach the limits of the list, scrolling will stop. Choose **Vertical**, **Horizontal**, or **Both** to enable wrap-around scrolling in those directions.
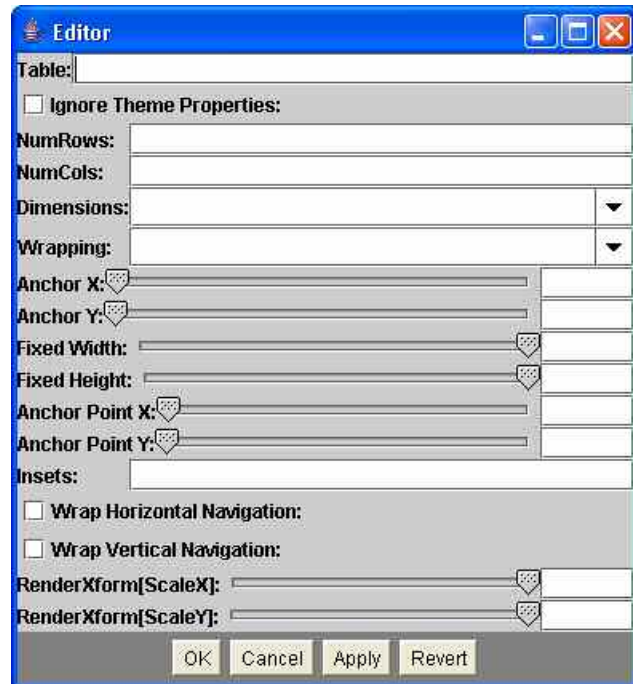
TableComponent Widget

TableComponent widgets are used with Table widgets as the table's constituent parts.

1-Dimensional Tables

To create a 1-dimensional table, place an action chain as the parent of the Table widget, with that action chain creating the data that populates the table. For the Table widget, set its Dimension property to use either **Vertical** or **Horizonatal**. A TableComponent of type **Cell** is placed as the direct child of the Table widget. The name of the TableComponent will become the local variable that is used to access the data element in the array that populates the Table.

2-Dimensional Tables

To create a 2-dimensional table, add a Table component with its Dimension property set to **Both**. 2-D tables require three TableComponents, one each for **Cell**, **RowHeader**, and **ColHeader**. Place action chains between the Table widget and its TableComponent widgets, with each action chain creating the data array for its child TableComponent. As for 1-D tables, the names of the TableComponent widgets are the local variables used to access the data elements for each of the three arrays populating the cells, row header, and column header.

TableComponent Widget Properties

The properties dialog for a TableComponent is shown to the right.

The first entry is the **name** of the widget. As discussed above, the name of the TableComponent is used as the local variable to access the data element in the data array that populates the table.

Most of the remaining properties were already covered previously; see Properties Common to Many Widgets. However, there is one more
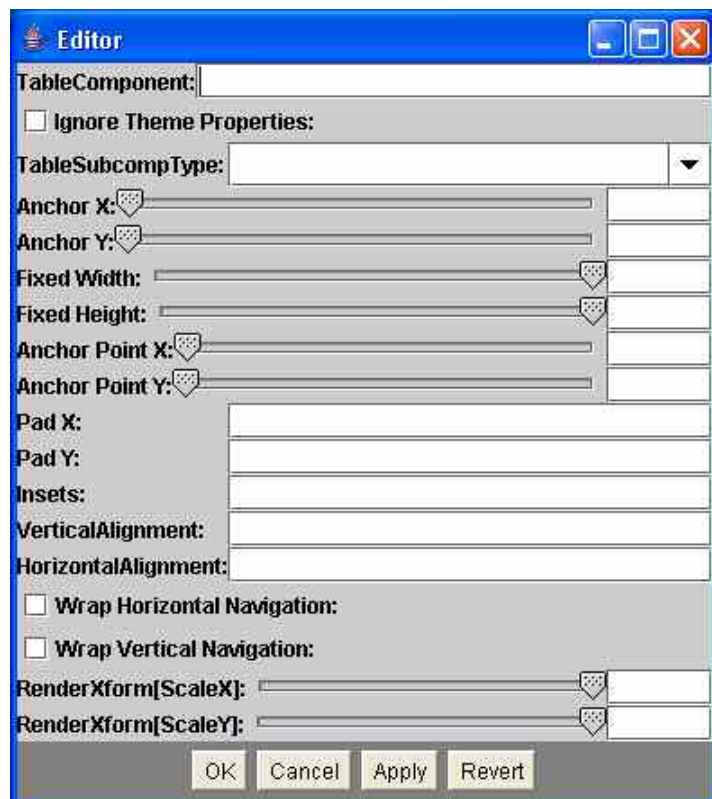
property that is specific to TableComponent widgets.

TableSubcompType is used to specify what type of table component the widget is to be. (Note that the **Nook** and **EmptyTable** choices are not currently used.) Every table must have a **Cell** component. These are the main data cells of any table, regardless of whether it is 1- or 2-dimensional. **RowHeader** and **ColHeader** are only used for 2-dimensional tables.

Text Widget

Text widgets simply display text inside its UI display area. There are two ways to specify what text will be displayed by the Text widget:

1. Set the name of the widget to the text to be displayed and do not place the Text widget under an action widget. (If there is no parent Action widget feeding data to the Text widget, then the name of the Text widget is the text to be displayed.)

2. Place the Text widget as the child of an Action widget. The result of the parent Action widget will be converted to a string before being displayed.

Tip: To create a blank line in some lines of text to be displayed, leave a space after the newline character in the Action widget: "\n" (without a space) will not result in an added blank line, since there is nothing to be displayed on the new line, but "\n " (with a space after the 'n') will add the blank line to the display.

Note that the font and colors used to display the text is set in the Theme that applies to the Text widget.

Text Widget Properties

The properties dialog for a Text widget is shown to the right.

The first entry is the **name** of the widget. As mentioned above, it is used as the text to be displayed if there is no parent Action widget feeding data to the Text widget.

Many of the remaining properties were already covered previously; see Properties Common to Many Widgets. Additional properties are as follows:
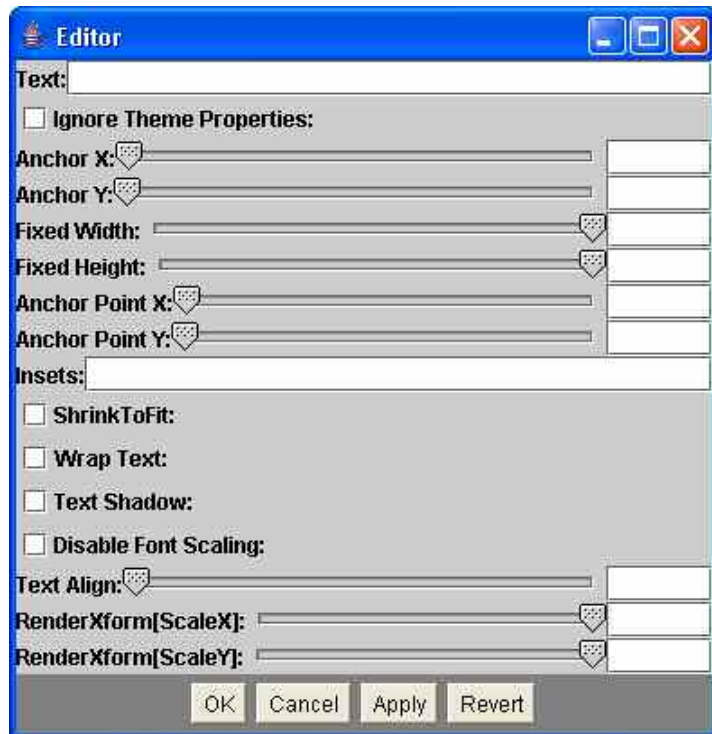
Insets works similarly to what is described in the above-referenced section, except that it affects the space placed around the text rather than child UI elements.

ShrinkToFit – Check this option to have SageTV automatically shrink the text to fit within the allotted display area.

Wrap Text – This option controls whether text that is too wide to fit into the Text widget's display area's width will wrap the text to the next line or simply cut off the remaining text.

Text Shadow – Choose whether the text should be displayed with a shadow or not.

Disable Font Scaling – If checked, the text will be displayed using the font point size specified in the applicable Theme widget instead of scaling the font size in relation to the size of the SageTV window. (The point size will remain the same no matter how large or small the SageTV window is.)

Text Align – This setting is the percent (0.0 – 1.0) position within the Text widget's total width where the text will be aligned. A value of 0.5 centers the text within the horizontal space allocated to the widget's UI display area. **Note:** This differs in comparison to using **Anchor X** + **Anchor Point X** in that those properties affect where the entire Text widget's display area is located within its parent's display area, while **Text Align** only affects the text's position within its own display area.

Image Widget

Image widgets simply display an image and/or a pressed image inside its UI display area. The image to be displayed may be specified by:

1. Set the **Image Source File** property for the widget to the filename of the image to be displayed. (See below.) This property is used if there is no parent action chain that passes an image to the Image widget.

2. Place the Image widget as the child of an Action chain that passes an image to the Image widget. An image may be passed as the result of a call to LoadImage(path), as a MediaFile if it is a Picture, as a string containing the path to the image, as a java.awt.image.BufferedImage object, or as a local variable that contains an image type. Note: the result of the parent action chain does not affect the pressed version of the image.

The Image widget can display .PNG, .JPG, or .GIF images. Java 1.5 will be able to also display .BMP files.

When an image is clicked with a mouse, it can either issue the SageTV command specified in the **Fire User Event** property or, if that property is blank, it will cause its child Action chain to be executed.

Image Widget Properties

The properties dialog for an Image widget is shown to the right.

The first entry is the **name** of the widget. The widget name is only for tracking the various Image widgets.

Many of the remaining properties were already covered previously; see Properties Common to Many Widgets. Additional properties are as follows:
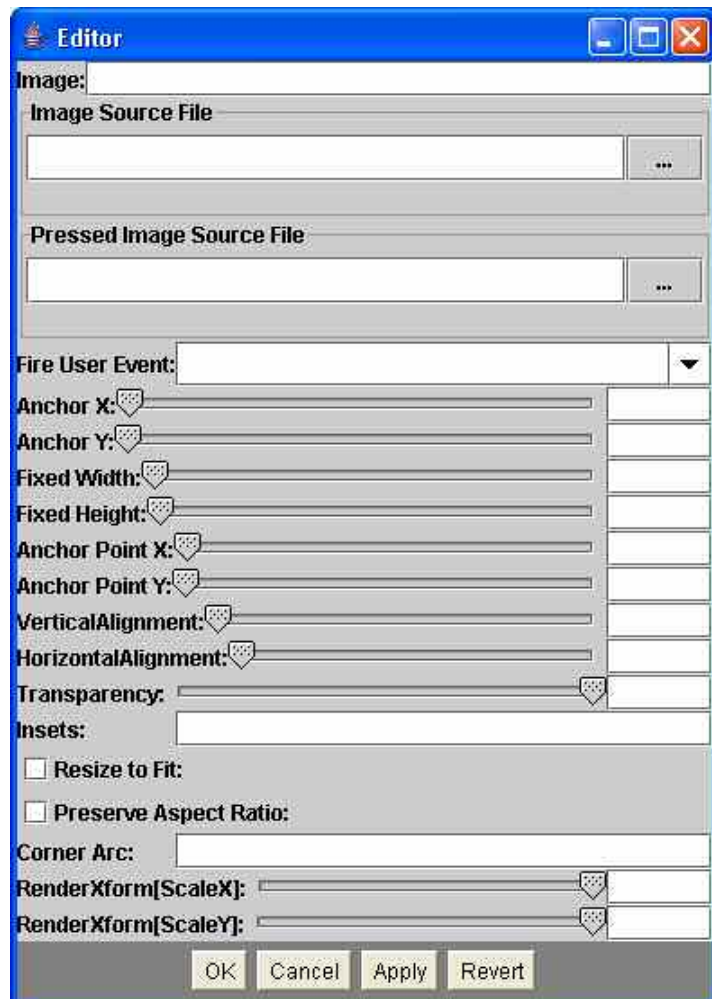
Image Source File – This is the normally displayed image, when it is not pressed. The image name specified here can be to resource paths (for images in the .jar file), partial file paths, absolute file paths and URLs (http and ftp are both supported). A dynamic local variable or attribute can be used to name the image to be loaded: set the variable equal to the name of the image and its path, as a string, then use "=LocalVarName" as the property value (without the quotes, of course).

Pressed Image Source File – This is the image displayed while a mouse button is clicked on the Image widget and is held down. The image name is specified in the properties dialog as above. (The pressed version of the image can only be specified within the properties dialog.)

Fire User Event – If an event is selected from the drop-down list, then clicking on the Image with a mouse will cause that SageTV command to be issued, just as if the SageCommand("CommandName") API function was called.

Transparency – Adjust the slider to determine the transparency level of the image, where 0.0 is fully transparent and 1.0 is fully opaque.

Insets works similarly to what is described in the above-referenced section, except that, as with Text widgets, it affects the space placed around the image rather than child UI elements.

Resize to Fit – Check this option to have SageTV resize the image to fit with the widget's display area.

Preserve Aspect Ratio – Choose this option to display the image at its original aspect ratio as it gets resized, so that the image does not appear to stretch in one dimension or the other.

Corner Arc – Use this setting to display the image cropped inside a rectangle with rounded corners. Use an integer value to specify the radius of the circle used to mask the corner of the image.

TextInput Widget

TextInput widgets are used to accept alphanumeric input, where the input gets stored as a string in a variable with the name of the TextInput widget. To access the input string, use an Attribute with the same name as the TextInput widget.

TextInput widgets are not focusable, so a Select Listener will need to be used to handle using Enter to accept input.

Note that the font and colors used to display the text is set in the Theme that applies to the TextInput widget. To prevent the input string from being displayed, use the **Hide Text** property, below.

TextInput Widget Properties

The properties dialog for a Text widget is shown to the right.

The first entry is the **name** of the widget. As mentioned above, it is the name of the variable used to hold the input string.

Many of the remaining properties were already covered previously; see Properties Common to Many Widgets. Additional properties are as follows:
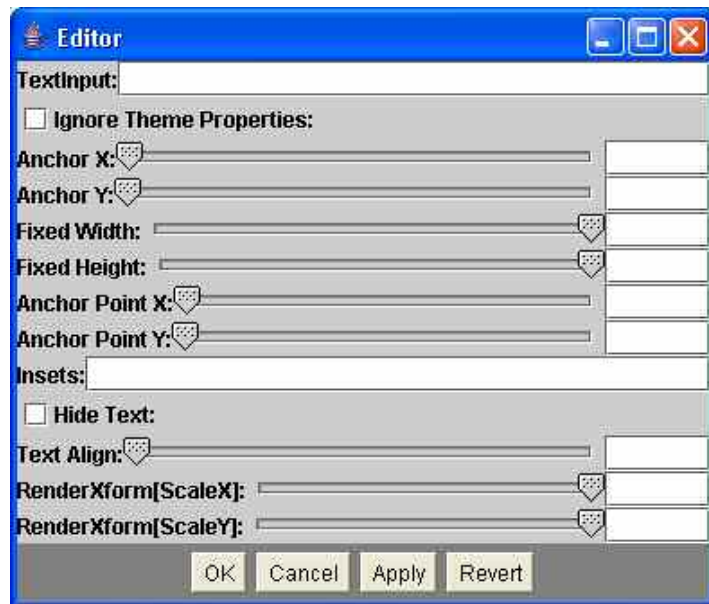
Insets – Like the Text widget, this works similarly to what is described in the above-referenced "common" section, except that it affects the space placed around the text rather than child UI elements.

Hide Text – Check this option to replace each entered character with an asterisk (*) instead of showing the entered character. This may be useful when entering passwords.

Text Align – This setting is the percent (0.0 – 1.0) position within the TextInput widget's total width where the text will be aligned. A value of 0.5 centers the text within the horizontal space allocated to the widget's UI display area. **Note:** This differs in comparison to using **Anchor X** + **Anchor Point X** in that those properties affect where the entire TextInput widget's display area is located within its parent's display area, while **Text Align** only affects the text's position within its own display area.

Video Widget

A Video widget is used to display video within a portion or all of the entire UI display area. Note: if a menu should always have video as its background, you may wish to simply check the **Video Background** option of the Menu widget.

The currently playing video is shown within the widget's display area.

If no video is playing when a Video widget newly shows up by either enabling the widget's visibility or by transitioning to a menu that has a visible Video widget, then SageTV will begin playback of a default video. The video shown could be a currently recording show or live TV on the last channel viewed.

Video Widget Properties

The Video widget properties dialog, shown to the right, simply consists of the widget's **name**, which has no special meaning, and some basic layout/positioning controls. See Properties Common to Many Widgets for a description of the layout properties.
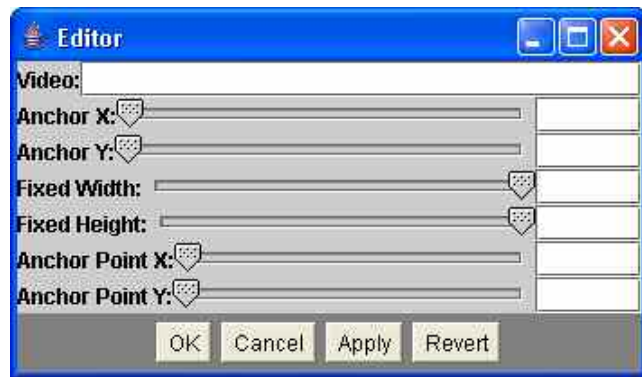
Shape Widget

A Shape widget is used to display basic shapes, either filled or outlined, with some level of alpha transparency. SageTV currently supports these shapes: **Square**, **Rectangle** (with rounded or square corners), **Circle**, and **Oval**. The shapes will be drawn within the allotted display area. See the widget's properties, below, for the settings that affect shape drawing.

Tip: beyond the fact that Shapes are drawn on the screen, they are not UI elements that a user can interact with. Unlike most other UI elements, Shapes may be included in loops to have multiple shapes drawn, depending on the details of each loop iteration. The playback OSD uses such loops for parts of the time progress bar, for example.

Shape Widget Properties

The properties dialog for a Shape widget is shown to the right.

The first entry is the **name** of the widget; it has no meaning other than a way to track the various Shape widgets in use.

Many of the remaining layout and sizing properties were already covered previously; see Properties Common to Many Widgets. Additional properties are as follows:

Shape Type – Choose what type of shape is to be drawn: **Square**, **Rectangle**, **Circle**, or **Oval**. Note: **Line** shapes are not fully supported at this time. For rounded squares or rectangles, select Square or Rectangle and also use the Corner Arc property.

Color – This property sets the shape's color. Select the color preview box to open a color selection dialog, or you may use a hex value or a dynamic expression to specify the color.

Alpha – This is the alpha blend value, used to determine the opacity level of the shape. Use the slider to set the alpha value from 0 (fully transparent) to 255 (fully opaque).

Color Fill – If checked, the shape will be drawn as a filled shape rather than only as an outline.
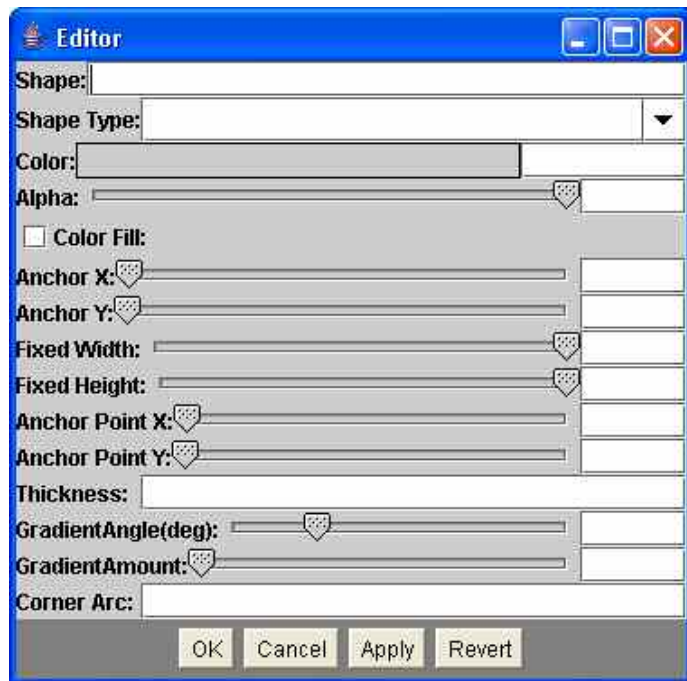
Thickness – If the shape is outlined rather than filled, this option determines how wide a line is drawn around the border of the shape. This value must be an integer specifying the width of the border in terms of the number of pixels wide the outline will be.

GradientAngle(deg) – If drawing the shape with a gradient color (see the next property), this is the angle (0 – 360) at which the gradient is drawn. Note: when using 3D Acceleration, angles that are multiples of 90 work the best, so it is recommended to always use values that are multiples of 90.

GradientAmount – This is the percent of the color to remove along the gradient angle. The value can range from 0.0 (remove no color; there will be no color gradient displayed) to 1.0 (remove all color along the gradient angle). The color along the gradient will progressively change from the selected color to a percent of black, so that if the **GradientAmount** is 1.0, then the shape will change from full color to complete black along the gradient. **Tip 1:** Enter negative decimal point values to change the gradient from a shift to black to a shift to white; however, you will need to enter larger negative numbers to see the color gradient. Try values in the range of -3.0 or -10.0. Such values are used when drawing the time progress bar in the playback OSD in the default version 3 STV. **Tip 2:** Draw multiple partially transparent different colored shapes on top of each other with varying gradient angles and amounts for more complex looking multi-hued shapes.

Corner Arc – Use this setting to display a **Square** or **Rectangle** cropped with rounded corners. Use an integer value to specify the radius of the circle used to mask the corner of the shape.

Attribute Widget

An Attribute widget is used to declare a variable in the local context for the parent Widget of the specified name. Its initial value is determined by evaluating the value property as an expression. The Value of the Attribute can be accessed through its name as a local variable in the tree extending below the Attribute widget's parent.

Attribute Widget Properties

The properties dialog for an Attribute widget is shown to the right.

The first entry is the **name** of the widget. The **name** is used as a variable name in the context of the widget's parent. (i.e.: in the tree under the parent.)

The **Value** property is used to initialize the Attribute. The expression entered here is evaluated and its result is used as the attribute's initial value. If nothing is entered, then Studio will report an error if **Notify On Errors** is enabled in the **Tools** menu.

Hook Widget

Hook widgets are executed whenever an event occurs in the SageTV core which creates a possibility that the UI may want to respond to that event. For example: before a menu is loaded and displayed, the core looks for a **BeforeMenuLoad** hook in the loading menu to see if that menu wishes to perform some function before the menu load continues.

Some hooks pass an argument as a local variable that the STV code can use. The above-mentioned **BeforeMenuLoad** hook passes a variable called **Reloaded** to tell whether the menu is being reloaded due to the use of the Back or Forward command. Also, some hooks allow a **ReturnValue** to be set before the hook's action chain completes to let the core know what action to take. The list of hooks, their arguments, and return values are listed later in this document; see 5) Hooks – The Complete List.

Hook Widget Properties

The properties dialog for a Hook widget is shown above. It consist solely of a drop-down list of available hooks to choose from.

Valid Widget Parent-Child Relationships

Not every type of widget can be a parent or child of every other type of widget. The valid relationships are as follows:

Valid child widget types for each parent widget type are marked with an X.	Menu	OptionsMenu	Panel	Theme	Action	Conditional	Branch	Listener	Item	Table	TableComponent	Text	Image	TextInput	Video	Shape	Attribute	Hook
Menu				X	X	X	X	X	X				X					X
OptionsMenu				X	X	X	X	X	X				X					X
Panel	X	X	X	X	X	X	X		X	X	X							
Theme	X	X	X						X	X	X	X	X	X				
Action	X	X	X			X	X	X	X	X			X					X
Conditional	X	X	X			X	X	X	X	X			X					X
Branch						X												
Listener	X	X	X	X					X	X	X	X	X		X			
Item	X	X	X	X	X	X	X		X	X	X							
Table	X	X	X	X	X	X	X		X									
TableComponent				X	X					X								
Text	X	X	X	X	X	X	X		X	X	X							
Image	X	X	X		X	X	X		X	X	X							
TextInput	X	X	X	X	X	X	X		X	X	X							
Video	X	X	X			X	X		X	X	X							
Shape	X	X	X			X	X		X	X	X	X	X	X				
Attribute	X	X	X						X	X	X	X	X	X				
Hook	X	X	X						X	X	X	X	X	X	X			

(Parent Widget across the top; Child Widget down the side.)

4) Attributes / Variables

Variable Context (Scope)

The variable context is a hierarchical context that goes down the UI tree for a given menu. Standard scoping rules apply in terms of which variable gets used (think {} in C/C++). Basically: a variable is accessible at the point it is initially defined and in all subsequent action chains that are children of the variable's creation point: an Attribute widget creates a variable that lives in its parent widget's context, while a variable created via an assignment expression on an Action widget lives in the context that Action is executing in (which usually corresponds to a UI Widget).

Variables may be set when defined in an Attribute widget, with a simple assignment action (example: var = 3), or via two API function calls:

- **AddGlobalContext** – The global context is the highest context which is never cleared or reset. A variable defined in the global context may be accessed at any point in an STV's tree.

- **AddStaticContext** – The static context gets copied into the new Menu's variable context (at the menu level) when a menu transition occurs, and then the static context is cleared. Use this as a way to pass a variable to the next menu without keeping it in the global variable context. If the same variable is to be transferred to a third menu, it would have to be added to the static context again before that menu transition occurs. (i.e. to show the Detailed Info for an Airing, you pass an Airing in the static context to the Detailed Info menu.)

When variables are accessed; if they don't appear in any context, then SageTV will create a new variable with that name in the current context and initialize that value to null.

The Attribute widget is used for initializing the variable AND for establishing it's scoping. Otherwise, the variable is auto-declared in the context where it is used....which means they might not be the same variable if it is used in multiple parts of a menu....but this problem is solved by having an attribute in a higher level context that is common between the different widgets that access the variable. So, if two separate sections of a tree are to have access to the same variable, make sure that it is initialized at a point which is a parent common to both places the variable is accessed. Again: it may be initialized either as a simple assignment on an Action widget, or as an Attribute widget. **Note:** It is often best to declare a variable via an Attribute widget, since this is more clear to others looking at the code than when auto-declaration is used.

How to Access Variables for the UI Element Currently in Focus

At times, it may be useful to know what variables are in use for the currently focused UI element while in some other section of the STV that does not share the UI element's variable context. Example: when displaying the show information for an airing highlighted in the LiveTV Guide, there has to be a way to know what show is highlighted, since the code displaying the show information is actually under a completely different menu, accessed via a theme. In such a case, use the **GetFocusContext()** API call to gain access to the variables in the context of the UI element in focus. This API call will copy all variables from the focus context into the current context except for variables that are shared between the two contexts at a higher level.

SageTV's Built-In Variables

In addition to creating and using your own variables, SageTV sets certain variables which you may access. These variables only exist in certain circumstances, as described below:

Predefined Local Variables

Certain variables are either set by SageTV for use in the STV code, or set in the STV code for use by SageTV:

- **this** – The results of an expression on any Action widget is stored in a variable named '**this**'. The '**this**' variable is only accessible in an expression on the very next widget, and its contents will be replaced by the results of that next expression.

- **DefaultFocus** – Declare an Attribute named **DefaultFocus** as a child of a focusable UI element to tell SageTV which focusable UI element is to receive the default focus for a Menu or OptionsMenu. The element to receive the default focus should be set to **true**, or should have its value set to an expression that can be evaluated to **true**. For example: an OptionsMenu that contains 3 options could default to the currently selected option when that menu appears. **Note**: The same effect can also be achieved using the MenuNeedsDefaultFocus() hook.

- **Focused** – Access this variable in a child action of a focusable UI element to determine if that element, or any recursive UI parent of that element, currently has focus. Example: The theme for Item widgets will often have a Conditional widget to check for **Focused** being **true**, and will draw that item in a highlighted manner to visually indicate focus.

- **TableRow, TableCol** – For tables, these are the index numbers of the current row or column. The values use a 1-based index, so there is no row or column zero.

Example: TableRow is used to display a number next to each entry in the SageTV Recordings list in the default STV.

⚘ **IsFirstPage, IsFirstHPage, IsFirstVPage** – These variables are true when at the first horizontal or vertical page in a table or scrollable UI element. Note: for a 2-dimensional table, **IsFirstPage** is true if both **IsFirstHPage** and **IsFirstVPage** are true.

⚘ **IsLastPage, IsLastHPage, IsLastVPage** – These variables are true when at the last horizontal or vertical page in a table or scrollable UI element. Note: for a 2-dimensional table, **IsLastPage** is true if both **IsLastHPage** and **IsLastVPage** are true.

⚘ **NumPages** – This is the number of pages in a table or scrollable UI element.

⚘ **NumRows, NumCols** – For tables only, these are the overall number of rows and columns in the table; i.e.: the number of items in the row or column, not the number of rows or columns displayed.

Listeners That Set Local Variables

The non-command listeners use the name of the Listener widget as the local variable holding details about the data input or event. If no name has been given to the listener, that data normally accessed via the listener's name can be accessed by using the '**this**' variable. Some of these listeners also set additional local variables. The listeners and their variables are:

⚘ **MouseClick** – This listener is used whenever a mouse click event occurs. The local variables available are:
   1) The name of the listener, or '**this**' – The number of the mouse button that was clicked. A value of **1** indicates the left button, while a value of **2** indicates the middle mouse button. (The right button is normally associated with the Options command, by default; however, that can be overridden in the properties file.)
   2) **ClickCount** – The number of times the button was pressed. This can be used to distinguish between single and double clicks.
   3) **X, Y** – These two variables are integers representing the (x,y) coordinates of the mouse click, in pixels.
   4) **RelativeX, RelativeY** – The values range from 0.0 to 1.0 and represent the relative point where the mouse was clicked. These values are relative to the size of the UI component in which the mouse click occurred.

⚘ **MouseDrag** – This listener is used whenever a mouse click-and-drag event occurs. The local variables available are:
   1) The name of the listener, or '**this**' – The number of the mouse button that was held while dragging. A value of **1** indicates the left button, while a

value of **2** indicates the middle mouse button. (The right button is normally associated with the Options command, by default; however, that can be overridden in the properties file.)

2) **X, Y** – These two variables are integers representing the (x,y) coordinates of the mouse pointer's position during the mouse drag, in pixels.

3) **RelativeX, RelativeY** – The values range from 0.0 to 1.0 and represent the mouse pointer's relative position during the mouse drag. These values are relative to the size of the UI component in which the mouse click occurred.

☺ **RawKeyboard** – Any time keyboard input is encountered, this listener is called, before the input is translated to any possible SageTV command it may be associated with. The local variables available are:

1) The name of the listener, or '**this'** – The key entered.

2) **KeyCode** – The key code of the key pressed, as defined in java.awt.event.KeyEvent.

3) **KeyModifiers** – The codes for any modifiers that were pressed along with the key, such as Ctrl, Alt & Shift as defined in java.awt.event.InputEvent.

4) **KeyChar** – The character entered, translating the key entered plus any modifiers, or a blank string if there is no corresponding character.

☺ **RawInfrared** – Any time infrared input is received, this listener is called, before the input is translated to any possible SageTV command it may be associated with. The local variables available are:

1) The name of the listener, or '**this'** – The infrared code received.

NOTE: If you're using a RawKeyboard or RawInfrared listener then they will intercept all respective events. In order to continue propagation of an event, you must use the PassiveListen() API call.

☺ **Numbers** – Any time a number key is pressed, this listener is called, before the input is translated to any possible SageTV command it may be associated with. The local variables available are:

1) The name of the listener, or '**this'** – The number entered, 0 – 9.

Special Widget Names

SageTV looks for and uses special widget names in certain circumstances:

☺ **Menus** – SageTV expects to find these Menu widget available for certain uses:

1) **Main Menu** – The Main Menu is the initial menu that SageTV loads at startup or when awoken from sleeping.

2) **Screen Saver** – This Menu is the one loaded when the SageTV screen saver is activated.

3) **Server Connection Lost** – When the client loses the connection to the SageTV server, it loads and displays this menu.

⊙ **Themes** – SageTV recognizes one specially named Theme widget:
1) **Global** – The theme named "**Global**" is the global theme that applies to the entire STV. It can be placed anywhere in the STV tree. If there is more than one theme named "Global", then the first one found will be used as the global theme. (The "first one found" may not be the one you expect, so it is best to only have one.)

5) Hooks – The Complete List

The available hooks, when they are called, their arguments, and any expected return values are all listed below. Any names listed in parentheses after the hook's name are arguments that the hook passes to the hook's action chain. The arguments are accessed as local variables using the name shown in the parentheses. For more information on the argument types, consult the SageTV API documentation at:
http://download.sage.tv/api/index.html

FilePlaybackFinished(MediaFile)

Called upon the **first** completion of a file being played; it is not called again if the user returns to the playback screen while the file is still loaded, rewinds a bit, and continues playing through to the end again. This hook will also not be called when SageTV is capable of automatically switching to the next file for playback (i.e. during playlists, music and live TV); but that behavior can be overridden with a property setting. Parameters:

1. **MediaFile** is the media currently being played/watched.

MediaPlayerFileLoadComplete(MediaFile, boolean FullyLoaded)

Called at two different times as the media player finishes its stages of loading a file. Parameters:

1. **MediaFile** is the file being loaded.
2. **FullyLoaded** will be one of two values, depending on what part of the load has been completed:
   a. **false** – The file is not yet fully loaded and ready to play yet, but the media's info is available.
   b. **true** – The file has completely finished loading and is now playing.

MediaPlayerError(String ErrorCategory, String ErrorDetails)

Called when an error occurs in the system that the user should be notified of. Parameters:

1. **ErrorCategory** is the category of the error.
2. **ErrorDetails** is a string describing the error.

RequestToExceedParentalRestrictions(Airing, String LimitsExceeded)

Called when an attempt is made to watch a recording that is restricted by the Parental Controls settings. Parameters:

1. **Airing** is the airing to be watched.
2. **LimitsExceeded** is a string listing the restrictions that have been exceeded..

When processing this hook, you will need to let SageTV know whether viewing the airing is allowed, which is done by setting this local variable:

☮ **ReturnValue** – Set to **true** if the specified airing may be watched.

RecordRequestScheduleConflict(Airing RequestedRecord, java.util.Collection ConflictingRecords)

Called when a user is attempting to add a manual recording to the recording schedule which conflicts with an existing scheduled recording. Parameters:

1. **RequestedRecord** is the airing being added to the recording schedule.
2. **ConflictingRecords** is an array of airings that the newly scheduled airing conflicts with.

When processing this hook, you will need to let SageTV know whether to override the conflict and add the requested airing to the schedule, which is done by setting this local variable:

☮ **ReturnValue** – Set to **true** if the requested recording is to be added to the schedule. In addition, you will need to cancel one of the scheduled recordings in the **ConflictingRecords** array in order to make room for the new recording. Note: this hook may be called multiple times in succession if the conflict cannot be resolved by only removing a single scheduled recording. After one conflict is removed, if there are still conflicts, this hook will be called again.

RecordRequestLiveConflict(Airing RequestedRecord, Airing ConflictingRecord)

Called when a user is attempting to add a manual recording that is currently airing to the recording schedule which conflicts with another recording. Parameters:

1. **RequestedRecord** is the airing being added to the recording schedule.
2. **ConflictingRecord** is the airing that the newly scheduled airing conflicts with.

When processing this hook, you will need to let SageTV know whether to override the conflict and add the requested airing to the schedule, which is done by setting this local variable:

☺ **ReturnValue** – Set to **true** if the requested recording is to be added to the schedule and to cancel the conflicting recording.

WatchRequestConflict(Airing RequestedWatch, Airing ConflictingRecord)

Called when an attempt to watch live TV conflicts with a recording that is already in progress. Parameters:

1. **RequestedWatch** is the airing the user is asking to watch.
2. **ConflictingRecord** is the airing that the airing to be watched conflicts with.

When processing this hook, you will need to let SageTV know whether to override the conflicting recording and watch the requested airing instead, which is done by setting this local variable:

☺ **ReturnValue** – Set to **true** if the requested airing is to be watched and to cancel the conflicting recording.

DenyChannelChangeToRecord(Airing AiringToRecord)

Called when watching live TV and SageTV is about to start a scheduled recording. The user can choose to deny the channel change and cancel the scheduled recording. Parameters:

1. **AiringToRecord** is the airing about to be recorded.

When processing this hook, you will need to let SageTV know whether to deny the channel change, which is done by setting this local variable:

☺ **ReturnValue** – Set to **true** to instruct SageTV to NOT change the channel. Live TV will continue and the scheduled recording will be cancelled.

InactivityTimeout()

Called after a period of no user activity.

NewUnresolvedSchedulingConflicts()

Called when SageTV discovers new unresolved conflicts in the recording schedule.

MediaPlayerPlayStateChanged()

Called whenever the media player's play state has changed.

MediaPlayerSeekCompleted()

Called after SageTV has finished changing to a new playback point after a seek request.

BeforeMenuLoad(boolean Reloaded)

Called during a menu transition, before the new menu has been loaded and displayed. You can initialize some variables here or even decide whether to jump to another menu instead. Note that the menu loading process stops while this hook is in progress; the menu load (or a new menu transition) will continue when the hook's action chain completes. Parameters:

1. **Reloaded** is **true** is the menu was reloaded due to the use of the **Back** or **Forward** command.

AfterMenuLoad(boolean Reloaded)

Called during a menu transition, after the new menu has been loaded and displayed. Note that the menu continues functioning while this hook is in progress. Parameters:

1. **Reloaded** is **true** is the menu was reloaded, as mentioned for the previous hook.

BeforeMenuUnload()

Called during a menu transition, before the current menu has been unloaded. Note that the menu does not exit until this hook's action chain completes.

MenuNeedsDefaultFocus(boolean Reloaded)

Called whenever SageTV needs to determine which item is about to receive the default focus. The default focus can then be overridden, such as with a call to the **SetFocusForVariable** API function. Parameters:

1. **Reloaded** is **true** is the menu was reloaded due to the use of the **Back** or **Forward** command.

RecordingScheduleChanged()

Called after SageTV has updated its recording schedule.

RenderingStarted()

Called when a UI component is about to start rendering. You can update variables in the action chain of this hook, but you should not include UI widgets in the tree or call UI-affecting API functions such as Refresh(). Since SageTV is in the process of updating the display of its UI, any processing done in this hook should be completed quickly. This is mainly used for dealing with animations and incrementing associated counters or timers.

FocusGained()

Called when a focusable UI component (items, panels, tables) has gained focus.

FocusLost()

Called when a focusable UI component (items, panels, tables) has lost focus.

STVImported(Widget[ ] ExistingWidgets, Widget[ ] ImportedWidgets)

Called when another STV file has been imported via the **ImportSTVFile** API function. Parameters:

1. **ExistingWidgets** – An array of the current STV's widgets.
2. **ImportedWidgets** – An array of the widgets imported from the new STV file.

MediaFilesImported(MediaFile[ ] NewMediaFiles)

Called when SageTV has found new media files in its import directories. Parameters:

1. **NewMediaFiles** – An array of the newly discovered media files.

StorageDeviceAdded(java.io.File DevicePath)

Called when SageTV has discovered that a new storage device has been added to the system. This can be used to have SageTV automatically respond when a user inserts a memory card from a camera, for example, or even connects a removable hard drive. Parameters:

1. **DevicePath** – The path to the root directory of the new storage device.

ApplicationStarted()

Called when SageTV is being started, just before the first menu is shown, after SageTV has been initialized.

Important Notes:

  ☣ **Do not use any code that attempts to show any UI elements. This hook is intended for data manipulation only.**

  ☣ **It is recommended that you do not use this hook unless you have a very good reason to do so and you fully understand why/how you are about to use this hook.**

ApplicationExiting()

Called when SageTV is exiting; it is the first thing SageTV does when starting the shutdown process.

Important Notes:

  ☣ **Do not use any code that attempts to show any UI elements. This hook is intended for data manipulation only.**

  ☣ **It is recommended that you do not use this hook unless you have a very good reason to do so and you fully understand why/how you are about to use this hook.**

6) The Studio Interface

User Interaction

Interaction with the Studio, like most applications, is done via mouse and keyboard input:

Using a Mouse

The mouse may be used to easily create, select, move, or copy Widgets and may be used as follows:

- ☯ **Left click** – Select a single widget
- ☯ **Shift+Left click** – Select all widgets between currently selected widget & newly selected widget
- ☯ **Ctrl+Left click** – Add an additional single widget to the current selection
- ☯ **Right click** – Show pop-up options menu
- ☯ **Double Left click** – Either:
  - A) Show/hide children widgets, or
  - B) Jump to the primary reference of a reference widget
- ☯ **Drag & drop a widget from the left-hand toolbar** – Add a new widget as a child of the destination widget
- ☯ **Drag & drop an existing widget** – Add a reference to the selected widget as a child of the destination widget
- ☯ **Ctrl+drag & drop an existing widget** – Copy the selected widget to be a child of the destination widget
- ☯ **Ctrl+Shift+drag & drop an existing widget** – Move the selected widget to be a child of the destination widget

Using a Keyboard

The keyboard also offers full control:

- ☯ **Up arrow** – Highlight widget directly above the current widget
- ☯ **Down arrow** – Highlight widget directly below the current widget
- ☯ **letter key** – Jump to the next widget that starts with that letter
- ☯ **Right arrow** – Show the widget's children
- ☯ **Left arrow** – Hide the widget's children
- ☯ **Ctrl+E** – Expand children
- ☯ **Ctrl+Shift+E** - Collapse all nodes
- ☯ **Ctrl+F** – Display the **Find** dialog, where you can enter text to search for; text searching is case sensitive, so "WidgetName" is not the same as "widgetname"
- ☯ **Ctrl+P** – Show the right-click pop-up options menu

- **F2** – Rename selected widget
- **F6** – Reload the menu that is currently displayed in SageTV
- **F5** – Load the selected menu widget in SageTV
- **Ctrl+A** – Select all
- **Delete** – Delete selected widgets
- **Ctrl+B** – Break selected widgets from parent
- **Ctrl+U** – Move selected widgets Up
- **Ctrl+D** – Move selected widgets Down
- **Ctrl+C** – Copy selected widgets to clipboard
- **Ctrl+X** – Cut selected widgets to clipboard
- **Ctrl+V** – Paste from clipboard
- **Ctrl+Shift+V** – Paste reference from clipboard
- **Ctrl+Z** – Undo
- **Escape** – Clear clipboard
- **Ctrl+L** – Show property editor for selected widgets
- **Ctrl+Return** – If editing a widget's properties, close the properties dialog and accept the changes

The Menus and Status Indicator

The Studio Menu Bar

The **File** menu contains these options:

- **New…** – Create a blank new STV.
- **Open…** – Open and load an existing STV, replacing the currently loaded STV. The filename of the loaded STV will switch to the file that is opened.
- **Open Xml…** – Open and load an STV previously saved in XML format, replacing the contents of the currently loaded STV. The STV filename will stay the same, but its contents will be replaced.
- **Save** – Save the current STV and rename the previously saved version of the STV to "<filename>.stv.bak".
- **Save As…** – Save the current STV to the filename specified. Studio will then have the newly-named STV loaded.
- **Save A Copy As…** – Save a copy of the current STV to the filename specified. Studio will continue to use the current STV, not the new filename.
- **Save A Copy As XML…** – Save a copy of the current STV in XML format to the filename specified. Studio will continue to use the current STV, not the new filename.
- **Import…** – Import an STV file into the current STV.
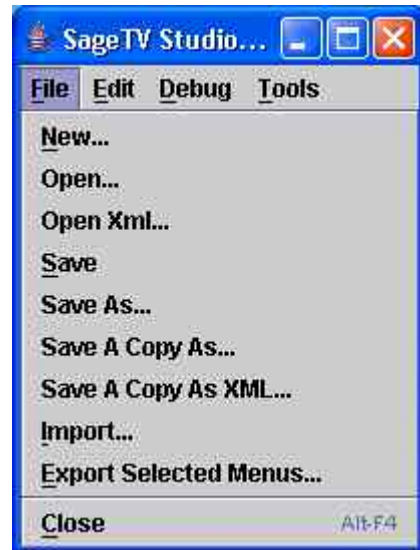
- ☮ **Export Selected Menus…** – Exports the selected menus to an STV file, which can then be imported into another STV.
- ☮ **Close…** – Close the Studio window. Studio itself will still be loaded in memory and will open again if you use the Customize command from within SageTV.

The **Edit** menu contains:

- ☮ **Undo** – Undoes the last operation EXCEPT for changes made within the Widget property editors.
- ☮ **Cut** – Cut selected widgets to clipboard.
- ☮ **Copy** – Copy selected widgets to clipboard.
- ☮ **Paste** – Paste from clipboard.
- ☮ **Paste Reference** – Paste reference from clipboard.
- ☮ **Delete** – Delete selected widgets.
- ☮ **Select All** – Select all visible widgets.
- ☮ **Find All…** – Searches for the specified text and selects all widgets where the text is found. The search is case sensitive, so searching for "WidgetName" is not the same as searching for "widgetname".

The **Debug** menu choices are:

- ☮ **Breakpoints…** – Opens the Breakpoints window, where all the current breakpoints are listed.
- ☮ **Tracer…** – Opens the Tracer window, where the enabled tracing items are listed.
- ☮ **UI Components...** – Opens the UI Components window, where all the active UI widgets for the current menu are displayed.
- ☮ **Pause** – Causes SageTV to enter Pause mode. SageTV will continue running, but will pause on the next Trace.
- ☮ **Resume** – Causes SageTV to resume normal execution mode.
- ☮ **Step** – Causes SageTV to step one level of execution.
- ☮ **Scroll on Trace** – If checked, Studio will scroll to each widget as execution continues.
- ☮ **Enable All Tracing** – Turns on tracing for all available trace types.
- ☮ **Disable All Tracing** – Turns off tracing for all trace types.
- ☮ **Trace <trace type>** – If checked, enables tracing for the named trace type.

The **Tools** menu choices are:

- ☺ **STV File Difference…** – Compares a selected STV file to the currently loaded STV and displays the differences in a comparison window.
- ☺ **Expression Evaluator…** – Evaluates the entered expression and displays the results. When in Debug mode, local variables may be used in the expression to be evaluated. (i.e. the expression is evaluated in the suspended context.)
- ☺ **Generate Translation Source…** – Generates the translation file to allow translating text to another language. For more information, see the online document: Language Translation/Localization of SageTV at http://sage.tv/2_papers/i18n.txt.
- ☺ **Consolidate Menus & "SYM:" Actions** – SageTV will consolidate the menus, replacing existing menus and references to those menus with links to larger menus that have the same names. This could be used to import a replacement menu that has more functionality than an existing version of that menu, then consolidate the menus to have the new, larger, menu replace the older one that had fewer children.
- ☺ **Notify On Errors** – Selecting this option toggles it on or off. If it is On, there will be a checkmark next to it in the menu and Studio will display a dialog box showing errors that are encountered as the UI is processed in real time.
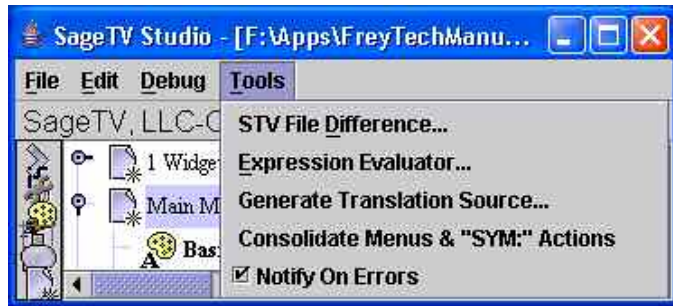
The Pop-up Options Menu

When a mouse right-click or Ctrl+p is issued, a **pop-up options menu** will appear for the selected widget. Available choices in the menu are shown in regular text, while unavailable options are grayed out. The menu items are:

- ☺ **New Child** – Adds a new child widget to the current widget. A roll-out menu is displayed, where the allowable child widgets for the current widget may be selected.
- ☺ **Expand All Nodes** – Displays the expanded tree of all children widgets for the entire STV.
- ☺ **Collapse All Nodes** – Collapses the entire tree display so that only the top-level widgets are visible.
- ☺ **Expand Children** – Displays the expanded tree of all children widgets for the currently selected widget.
- ☺ **Refresh Menu** – Reloads and refreshes the current menu shown in the SageTV window.

- **Break From Parent** – Breaks the Parent <-> Child relationship for the currently selected widget. If the selected child widget is a reference widget, the reference will simply be removed. If the selected child widget is not a reference, it will be moved to the top level of the STV tree.
- **Move Up** – If the selected widget is not the only child of its parent and not already at the top of the list, it will be moved up one position.
- **Move Down** – If the selected widget is not the only child of its parent and not already at the bottom of the list, it will be moved down one position.
- **Highlight References** – Highlights and selects all references for the currently selected widget.
- **Set as Primary Reference** – Makes the currently selected *italicized* reference to be the **bolded** primary reference. Only the primary reference widget can have its children expanded for viewing.
- **Add Breakpoint** – Adds a breakpoint for the currently highlighted widget.
- **Remove Breakpoint** – Removes the breakpoint for the currently highlighted widget, if there is a breakpoint at that widget.
- **Paste Properties** – Pastes the property values from the copied widget to the destination widget. Useful for when you want to copy a widget onto something that's already there without having to delete it.
- **Launch Menu** – Launches the currently highlighted menu in the SageTV window.
- **Evaluate Widget** – Evaluates the currently selected widget and shows the results in a pop-up dialog
- **Execute Widget Chain** – Executes the widget chain, starting at the selected widget.
- **Rename** – Enters edit mode for the name of the selected widget.
- **Delete** – Deletes the currently selected widget(s). If the widget's children were not yet deleted, they will be moved to the top level of the STV tree. Children that were references will simply have that reference removed.
- **Properties** – Opens the properties editing dialog for the currently selected widget(s).
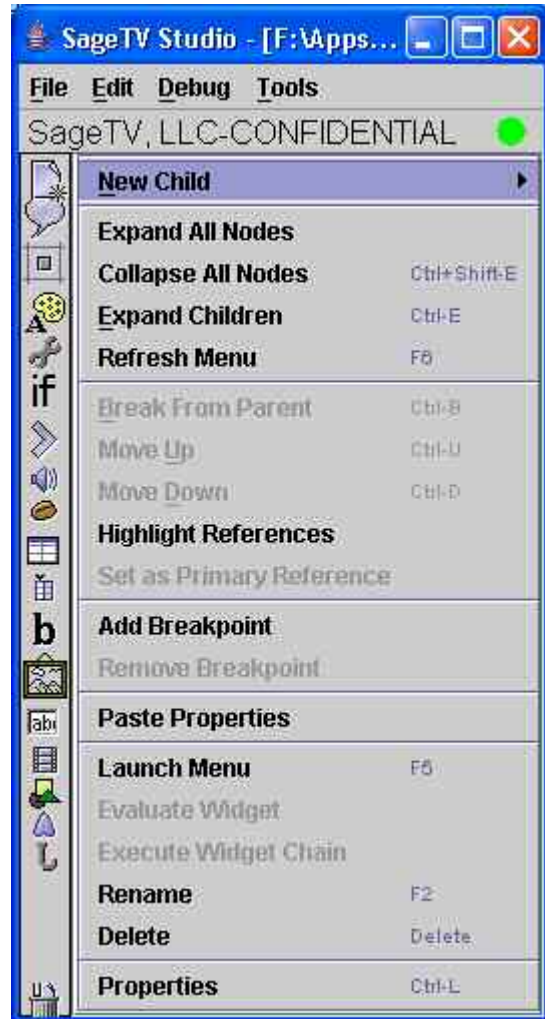
The Widget Bar

In addition to the pop-up option menu's **New Child** item, widgets may be added to the STV code by dragging the desired widget from the Widget Bar, located on the left side of the Studio display, as shown below. Simple click on the desired widget type, drag it to the code area, and drop it onto the widget where you want the new widget added as a child. The new widget will be added as the last child under the widget it was dragged onto.

The items on the Widget Bar are:

Menu – The top-level UI Widget. (see Menu Widget)

OptionsMenu – Provides 'pop-up' menus for SageTV. (see OptionsMenu Widget)

Panel – A rectangular UI element container. (see Panel Widget)

Theme – Used to apply a general appearance to a UI Widget hierarchy. (see Theme Widget)

Action – Actions are expressions that are executed as SageTV runs. (see Action Widget)

Conditional – Used to conditionally execute an action chain. (see Conditional Widget)

Branch – Used for multiple-option **Conditional** branching. (see Branch Widget)

Listener – Used to respond when the user issues a command. (see Listener Widget)

Item – A UI element that can have 'Focus' and can be selected by the user. (see Item Widget)

Table – Used to create dynamic, scrollable UI components. (see Table Widget)

TableComponent – Used inside a Table to specify the elements of the **Table**. (see TableComponent Widget)
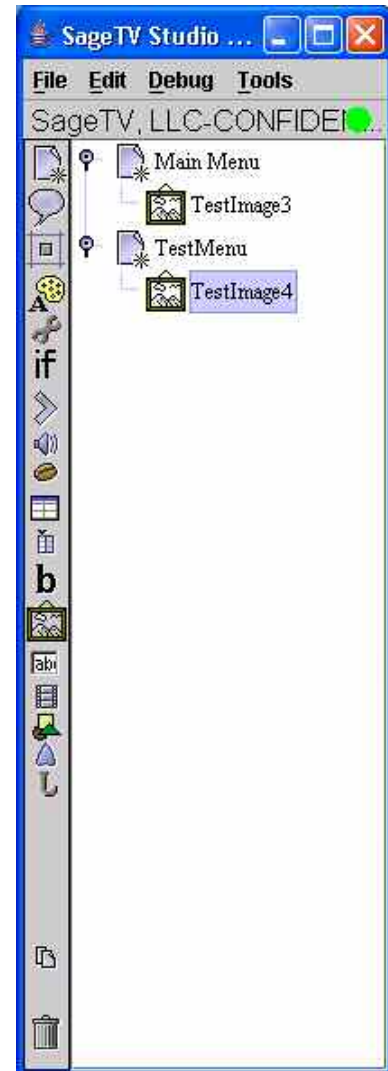
b **Text** – Used to display text. (see Text Widget)

Image – Used to display an image. (see Image Widget)

TextInput – Used for text entry. (see TextInput Widget)

Video – Used to display video. (see Video Widget)

Shape – Used to draw basic geometric shapes. (see Shape Widget)

Attribute – Creates a variable to store data for later reference. (see Attribute Widget)

Hook – Used like a callback system where the UI can respond to certain events that happen in the core. (see Hook Widget)

Duplicate – Drag a widget from the code display to this icon to duplicate it.

Delete – Drag a widget from the code display to this icon to delete it.

The "Running" Indicator

As seen in the image to the right, there is a status bar below the Studio menu bar. At the far right of the status bar is a colored dot indicating the running status of SageTV. Its color can be one of the following:

- **Green** – SageTV is running.

- **Yellow** – SageTV is currently running, but it will stop on the next operation encountered.

- **Red** – SageTV has stopped on a breakpoint.

Basic STV Editing

As shown by the menu bar items above, STV files can be loaded and saved with the Studio like you would any other file in an application. You can also import the contents of an STV file into the currently loaded one, as well as export a set of Widgets from a loaded .stv file into a new .stv file on disk.

What's not standard is that the .stv files are always being saved. Any modifications you make are more-or-less instantaneously saved to disk. The Save command does have a

purpose though, it will compress down the information stored in a loaded .stv file by compacting all of the incremental save information. (**NOTE:** This will change in the future to behave like standard Windows applications do in terms of loading and saving documents. When that change occurs, edits will be saved in a separate file for recovery purposes, while complete saves to the original STV file will not occur until such a command is issued by the user.)

If you make changes to a .stv file that you'd like to undo, you do have the ability to revert the .stv file back to its state before it was loaded. This is done by replacing the .stv file with the .stv.bak file which is created each time a .stv file is loaded or saved. The .stv.bak file contains the .stv file before it was last edited with the Studio. (The current .stv file is also copied to the .stv.bak file when the .stv file is saved via the Save command.)

There is also an individual Undo command which undoes the last operation EXCEPT for changes made within the Widget property editors.

Note: For more detailed examples of Studio usage, see 8) Studio Tutorials and Examples.

Widget Manipulation

Widget relationships and properties can be modified by using various drag & drop, copy & paste or other right-click menu options and keystrokes.

Adding Widgets

Keyboard: Highlight an existing widget where you wish to add a new child widget. Use Ctrl+P to show the Pop-up Options Menu, select New Child, then select the desired new widget. The new widget will become the last child of the existing widget. Note that only widgets that can be added as children of the highlighted widget will available.

Mouse: Left-click on the desired widget on the Widget Bar, drag it to the editing window, and drop it onto an existing widget, where the new widget will become the last child of the existing widget. Note that if the widget is not allowed to be a child of an existing widget, a red slashed-circle will be shown on top of the widget as it is dragged over that existing widget.

Removing Widgets

To remove widget(s), simply highlight the widget(s) to be removed and use the Delete key or the Delete command from the Pop-up Options Menu.

Care must be taken when deleting widgets, especially when those widgets have children. If a deleted widget had children, those child widgets may become orphan widgets in the top level of the Studio-displayed tree, serving no purpose. It is often best to delete widgets from the bottom of a tree branch towards the top. An exception is when deleting bolded widgets…

If a bolded widget is deleted, one of the other references to that widget will become the new Primary Reference. The child widgets will not become orphans, but will remain in-place under the new primary reference. **Caution: Be careful when deleting child widgets of one that is shown in bold, since such code changes will affect all references to that bolded widget.**

If an italicized widget is deleted, just that reference is removed. The Primary Reference widget and its children are not affected.

Moving and Copying Widgets

Child order: To change the order of the children of a parent widget, highlight a widget and use either Ctrl+U to move the child Up in the list of children, or Ctrl+D to move it Down. (Those commands are also available in the Pop-up Options Menu.)

Copying a widget: Select a widget using Ctrl+C. It will become outlined. Highlight the widget where you wish to copy it to. Use Ctrl+V to paste the widget as a copy. Using a mouse, drag the chosen widget and press the Ctrl key while dropping it to make a copy.

Create a reference to a widget: Select a widget using Ctrl+C. It will become outlined. Highlight the widget where you wish to create a reference to the selected widget. Use Shift+Ctrl+V to paste a reference to the selected widget. Using a mouse, drag the chosen widget and drop it to create a reference.

Moving a widget: Select a widget using Ctrl+X, to cut the widget. It will become outlined and will remain at its current position for now. Highlight the widget where you wish to move the selected widget. Use Ctrl+V to move the selected widget – it will be removed from its old location and become a child at the new location. Using a mouse, drag the chosen widget and press the Shift+Ctrl keys while dropping it to move the widget.

Undo: To undo an edit, use Ctrl+Z.

Editing Widgets

To quickly change the text shown next to a widget icon in the Studio tree, highlight the widget and press F2. The widget's title will become editable. Press return when done. Press Escape to cancel any changes.

To change any of a widget's properties, highlight the widget and press Ctrl+L to open that widget's properties dialog or just right click on the widget and select Properties. **Note:** The properties for multiple widgets of the same type may be edited all at once by selecting multiple widgets before launching the property editor.

See 3) Widget Details for details regarding all widget properties.

When done editing the widget properties, select the **OK** button, or see Properties Dialog Buttons for information regarding other choices.

Using Studio – A Beginning Tutorial

Note: This section is a very simple introduction to some basic Studio widget manipulation. For more detailed examples of Studio usage, see 8) Studio Tutorials and Examples.

1. Create a new .stv file using the File->New command.
2. To create new Widgets, click on their icon in the Widget Toolbar and drag it into the tree area of the Studio and drop it. Create a Menu and an Image.
3. You should now have 3 Widgets. A "Main Menu" Menu, an "Untitled" Menu and
   an "Untitled" Image. Note that new, untitled, widgets added to the top level are placed at the top of the tree; top level widgets that are named are listed in alphabetical order.
4. Right-click on the "Untitled' Menu and select Rename (or type F2). Type in: "TestMenu" and hit Enter.
5. Right-click on the "Untitled' Image and select Rename. Type in: "TestImage" and
   hit Enter.

6. Drag and drop the TestImage onto the TestMenu. TestImage is now a child of TestMenu (likewise TestMenu is a parent of TestImage)

7. Drag TestImage and drop it onto Main Menu. TestImage is now a child of BOTH Main Menu and TestMenu. IMPORTANT: When you do normal drag & drop in the Studio it CREATES a new parent-child relationship.

8. Whenever a name is *italicized* in the Studio, it indicates that it is a "Reference" to the actual Widget. If a name is **bolded**, then that Widget is a Primary Reference and has one or more references to it. (This will become more clear in a minute)

9. Right-click on the non-italicized **bolded** TestImage and select Rename. Type in:

"TestImage2" and hit Enter.

10. Right-click on the *italicized TestImage2* and select Rename. Type in:

"TestImage3" and hit Enter.

11. Right-click on the *italicized TestImage3* and select "Break From Parent".

"Break From Parent" will destroy the parent-child relationship between the selected object and the graphical parent in the tree of what was selected.

12. Click and drag TestImage3, hold down the Ctrl and the Shift key on the keyboard and then drop it onto Main Menu. When you press **Ctrl+Shift**, an 'arrow' sign should appear next to the cursor to indicate a **move**.

13. Click and drag TestImage3, hold down the Ctrl key on the keyboard and then drop it onto TestMenu. When you press the **Ctrl** key, a 'plus' sign should appear

next to the mouse cursor to indicate a **copy**.

14. Right click on the TestImage that is a child of TestMenu and select Rename. Type
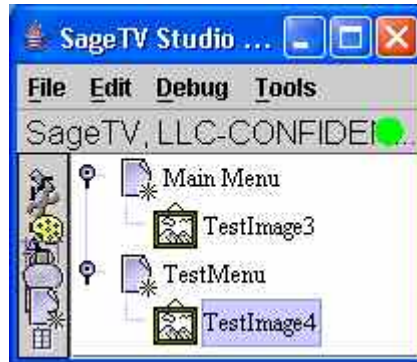
in: "TestImage4" and hit Enter. Note that only the copy was changed this time: the copied widget is not a reference to the one it was copied from; it is a separate widget now.

7) Using The Debugger

As briefly mentioned in The Studio Menu Bar, Studio has debugging capability, allowing the user to set breakpoints, step through individual lines of code, watch the Tracer information for certain events, and view the UI Component tree to see all the interface components for the current menu.

Breakpoints

To set or remove a breakpoint for a widget, simply select the widget and use the **Add Breakpoint** or **Remove Breakpoint** options available on the pop-up options menu, accessed with a right mouse click or Ctrl+P.

To see a list of current breakpoints, use the **Debug** menu's **Breakpoints…** item. Selecting that item will show the Breakpoints window, where all breakpoints will be listed. Click on any breakpoint to jump to that widget in Studio.

Code Tracer

Select **Tracer…** from the **Debug** menu to see the Tracer output window. As SageTV runs, any tracers selected in the **Debug** menu will display their results in this window. While tracing is active, the current widget will be highlighted in Studio if the **Scroll on Trace** option is checked in the **Debug** menu.

UI Components

Select **UI Components…** from the **Debug** menu to see the UI Component window. This window displays all of the UI widgets that are in use for the current menu. The UI components that are not actually displayed in the SageTV window have a yellow marker on the widget's icon. Right click in this window for some options for expanding the displayed widget tree and for highlighting the widget or its theme in the main Studio window.

Stepping Through Code

SageTV will halt execution when it encounters a widget where a breakpoint has been set. In addition, Studio can be set to halt execution by choosing the **Pause** item on the **Debug** menu – SageTV will continue running, but will break execution on the next operation.

Once execution has been halted, you may continue execution via these **<u>D</u>ebug** menu options: **Resume** (resume execution until the next break), or **Step** (step one level of execution).

See The "Running" Indicator for information about the current running status of SageTV.

Tip: While paused, you may wish to use the **<u>T</u>ools** menu's **Expression Evaluator…** to evaluate expressions using the current local variables (the current context).

8) Studio Tutorials and Examples

Several tutorials and examples are provided in this chapter as a way to help you become more familiar with Studio usage. An STV file containing the tutorials has been provided: **Studio_Tutorials.stv**. That STV can be loaded and experimented with while reading the tutorials in order to gain first hand experience.

The tutorials cover some of the basic Studio features, providing a base of knowledge about how the manipulation of widgets affects the SageTV user interface. The tutorial STV is a very basic file that does not provide a complete interface for SageTV. For more complex examples, check out the default SageTV STV (**SageTV3.STV**) or other custom STV files available online. Check our forums at http://forums.sagetv.com/forums/ for customizations that other users have created.

The examples are walk-throughs of sample edits that may be desired to change the functionality of the default STV, such as adding or moving menu items and so on.

Note: While following these tutorials in Studio, be sure to create a copy of Studio_Tutorials.stv and load that copy in Studio. That way, you can experiment with the copy without affecting the original, so that the original will always be available for reference if the experimental changes adversely affect the STV.

Similarly, when following the examples in Studio, make a copy of the default STV, SageTV3.STV, and edit that copy instead of the original.

If you do make a mistake and need the original STVs, they may be downloaded at:

⊕ **http://download.sage.tv/Studio_Tutorials.stv**

⊕ **http://download.sage.tv/SageTV3.stv**

Tutorial Set 1 – Basic Widget Manipulation

The first steps for this first tutorial will be to open Studio (see Starting Studio) and load the tutorial by using File -> Open menu option, where you can select the STV file to be loaded: **Studio_Tutorials.stv**. The Main Menu will now be active, but it is has no feature other than a short message announcing itself.

Find the menu used for this tutorial by typing Ctrl+F to open the **Find** dialog, then type **Tutorial 01** (use the exact case for each letter in the word Tutorial, since the search function is case-sensitive, and follow it with a space and the number one). Press Return. Studio will highlight the **Tutorial 01 – Basics** menu – if it isn't visible, scroll down until you see it.

After the menu widget titled **Tutorial 01 – Basics** is selected, press the Right arrow three times to expand that menu's tree a couple levels. (Or: instead of the Right arrow, click on the tree expansion indicator to the left of the menu widget icon, and then expand the "If false" line also.) You will see a series of panel widgets below the If statement.

Note: This menu just contains widgets for sample manipulation; it will not display anything useful in the SageTV window if it were to be loaded as the active menu. The "If false" statement prevents SageTV from trying to execute any code used for the tutorial in this menu – these sample widgets are not syntactically correct for code execution purposes.

Adding New Widgets

Highlight the "Add New Widgets" panel and notice that there is no expandable-tree indicator next to this panel, unlike the panels below it. If you use the Right arrow, the highlight will simply move to the panel below this one – if the panel had child widgets, the tree would have expanded further instead.

You can add new widgets to the "Add New Widgets" panel by dragging widgets from the widget bar to the left of the edit window. Drag an Action widget (🔧) to the panel and release the mouse – a new Action widget called "Untitled" will be added as a child of the panel. Continue adding a few more widgets as children of the panel or any of the new widgets. Note that some widgets may not be added as children of certain other widgets; when that is the case, you will not be able to drop the widget on top of the one where it is not allowed. See Valid Widget Parent-Child Relationships for a reference of which widget types may be children of other widget types.

When done, click on the down-pointing indicator to the left of the "Add New Widgets" panel icon to collapse the entire tree below that panel.

Rearranging Widget Order

Highlight the "Rearrange Widget Order" panel and press Ctrl+E to completely expand the tree containing its child widgets. You will see a "Parent Action" widget and a series of four numbered Action widgets below that parent. If the numbered child widgets were valid code, they would be executed in the order shown: 1, 2, 3, then 4. (Remember: these Action widget titles are for tutorial purposes only; they do not actually do anything.)

If you wish to change the order that widgets are executed when all the widgets are at the same tree level, simply highlight a widget among the ones to be re-ordered and use the Move Up (Ctrl+U) or Move Down (Ctrl+D) commands. The widget will move up or down once each time the command is issued. Move some of the numbered widgets up or down. Note that they cannot be moved above the top position or below the bottom position.

When done, collapse the "Rearrange Widget Order" panel.

Moving Widgets

Highlight the "Move Widgets" panel and press Ctrl+E to completely expand the tree containing its child widgets. You will see that "Child Action A" has a child widget with further children below that, while "Child Action B" has no children. Let's say that we wish to **move** A's children to be children of B.

Highlight "Action widget 1" and press Ctrl+X. (It should now be outlined when you highlight a different widget.) Highlight "Child Action B" and press Ctrl+V. The selected widget will be cut from its original location and added as a child of B. Note that all of the additional children were moved also, since they are children of the widget that was moved.

To undo this move, use Ctrl+Z.

To move a widget with the mouse, press and hold the Shift+Ctrl keys while dragging the widget to be moved, and drop it at its new location. Notice the small arrow next to the mouse cursor while pressing the Shift+Ctrl keys – that indicates that the drag & drop process will move the widget(s) being dragged.

When done experimenting with moving widgets, collapse the "Move Widgets" panel.

Copying Widgets

Highlight the "Copy Widgets" panel and press Ctrl+E to completely expand the tree containing its child widgets. You will see that "Child Action A" has a child widget with further children below that, while "Child Action B" and "Child Action C" have no children. Let's say that we wish to **copy** one or more of A's children to be children of B or C.

To copy a single widget: Highlight "Action widget 1" below "Child Action A" and press Ctrl+C. (It should now be outlined when you highlight a different widget.) Highlight "Child Action B" and press Ctrl+V. The selected widget will be copied from its original location and added as a child of B. Note that all of the additional children were NOT also copied; only a copy of the single selected widget was made. You could now edit that copied widget without affecting the original below A.

To copy multiple widgets: First, select multiple widgets to be copied. This can be done with the keyboard by highlighting "Action widget 1" below "Child Action A" and pressing the Shift key while pressing the Down arrow twice. (Or: press the Ctrl key while left-clicking on the widgets to be selected.) "Action widget 1", "-2", and "-3" should now all be highlighted. Press Ctrl+C to copy them to the clipboard. Highlight "Child Action C" and press Ctrl+V. The selected widgets will be copied from their original location and added as children of C, maintaining the original tree structure for widgets 1, 2, and 3. You could now edit those copied widgets without affecting the originals below A.

To undo a copy, use Ctrl+Z.

To copy widgets with the mouse, press and hold the Ctrl key while dragging the widget(s) to be copied, and drop them at the new location. Notice the small plus-sign next to the mouse cursor while pressing the Ctrl key – that indicates that the drag & drop process will copy the widget(s) being dragged.

When done experimenting with copying widgets, collapse the "Copy Widgets" panel.

Copy Widget Reference

Remember from Widget Relationships that all widget links are actually references, but that a single widget may have reference links to it from multiple parent widgets. As a reminder: when a widget has multiple references to it, the **Primary Reference** is shown **bolded**, while all other references are shown *italicized*.

Highlight the "Copy Widget Reference" panel and press Ctrl+E to completely expand the tree containing its child widgets. You will see that "Child Action A" has a child widget with further children below that, while "Child Action B" has no children. Let's say that we wish to create a reference to A's children below B.

Highlight "Action widget 1" and press Ctrl+C. (It should now be outlined when you highlight a different widget.) Highlight "Child Action B" and press Shift+Ctrl+V. The selected widget will be cut from its original location and added as a child of B. Note that the original widget is now **bolded**, while the widget placed below B is *italicized*. Editing "Action widget 1" will now change it in both places and its children still apply to both locations, even though they are shown only below the **bolded** primary reference. (Again, see Widget Relationships for more details.)

To undo a reference copy, use Ctrl+Z.

To copy a widget reference with the mouse, simply drag the widget to be copied and drop it at its new location. The default drag-and-drop action is to create a reference copy.

When done experimenting with copying widget references, collapse the "Copy Widget Reference" panel.

Deleting Widgets

As preparation for this tutorial, see Removing Widgets for an overview of widget removal. Note that parent widgets should not normally be removed before the child widgets have been removed, or they should at least be removed at the same time; otherwise, orphan widgets may be left in the top level of the tree displayed in Studio.

For this tutorial, completely expand the "Delete Widgets" panel. Note that the sample widget tree is similar to the previous tutorials, except that there is already a widget with multiple references ("Action widget 1"). We will be deleting widget(s), noting the results, then undoing the deletion to show what happens in various situations.

  ⚙ **Highlight "Action widget 3" and press the Delete key.** Note that the widget has been completely removed. That widget had no children and was safely removed without causing any side effects. The other widget at the same level, "Action widget 4", became the only child widget at that level. **Press Ctrl+Z to undo the deletion before continuing.**

  ⚙ **Highlight "Action widget 2" and press the Delete key.** That widget had two children that had not yet been deleted. In addition, those children were not referenced by any other widgets, so they no longer have any parent widgets. If you scroll the Studio window so that you can see the top level widgets starting with "A", you will see "Action widget 3" and "Action widget 4" as orphans in the top level of the tree. **Press Ctrl+Z to undo the deletion of "Action widget 2".** (You may need to expand the tree below "Action widget 1" again.) Note that the widgets are back in their original locations.

  ⚙ **Delete the *italicized* reference to "*Action widget 1*" that is below "Child Action B".** All this has done is to delete a secondary reference to "Action widget 1". No widget was actually deleted; only a reference to a widget was removed. However, note that the "Action widget 1" widget below "Child Action A" is no longer bolded, because there are no longer multiple references to that widget. **Undo the deletion of "*Action widget 1*".** Note that the *italicized* reference has been restored and the primary reference is **bolded** again.

  ⚙ **Finally, delete the bolded reference to "Action widget 1" that is below "Child Action A".** You have now removed the primary reference to "Action widget 1". As in the previous step, no widget was actually deleted; only a reference to a

widget was removed. When a primary reference to a widget is deleted, some other secondary reference becomes the new primary reference. In this case there was only one other reference. (You may now need to expand the tree below "Child Action B".) Note that the "Action widget 1" widget below "Child Action B" is no longer italicized, because it is no longer a secondary reference to that widget. It is not bolded either, since there are no longer multiple references to that widget. That widget and its children are simply fully accessible in that widget tree's only remaining location. **Undo the deletion of "Action widget 1".** Note that the deleted reference to "Action widget 1" has been restored, but it is now an *italicized* secondary reference to that widget. **Tip:** You can change which widget is the primary reference: right click on the *italicized* reference to "*Action widget 1*" and select **Set as Primary Reference**.

When done experimenting with deleting widgets, collapse the "Delete Widgets" panel.

Conclusion

This ends the **Basic Widget Manipulation** tutorial, where you learned how to add widgets, rearrange their order, move them, copy them, create references to widgets, and delete them, along with how to undo your previous commands. When done experimenting with any widgets in this menu, collapse the "Tutorial 01 - Basics" menu widget tree.

Tutorial Set 2 – Text Display

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 02 – Text Display". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window (from the theme, to be discussed in a later tutorial) and two sections with text. **Note:** The use of F5 shifted focus to the SageTV window when the menu loaded, so be sure to switch back to Studio to continue using it. You can use F6 to refresh the current menu once it has been loaded.

Before continuing, you may wish to review the Text Widget reference.

Static Text Display

The text in the top half of the SageTV window is displayed via the first text widget – the one w/o an action widget as its parent. Since this text widget has no parent action widget, the title of the widget is the text displayed in SageTV. Highlight the widget. Note that the widget's area in the SageTV window (the entire top half of the screen) is outlined in yellow.

While the widget is highlighted, press F2. The widget's display will change to a text editing box. You can change the text to say whatever you want and then press Return. Press F6 to update the SageTV window, but note that it shifts Windows focus to the SageTV window, so switch back to Studio to continue editing. (Ctrl+Z will undo the edit, but make sure Studio has focus or SageTV will be put into sleep mode, since Ctrl+Z defaults to the Power command in SageTV.)

Dynamic Text Display

The text in the bottom half of the SageTV window is displayed via the second text widget – the one with an action widget as its parent. Since this text widget has a parent action widget, the data passed to the text widget via the results of executing the parent action widget is the text displayed in SageTV. (The '\n' characters create a newline character in the text to be displayed.) Note that the action widget's results (a text string, in quotes) override the title of the text widget.

Highlight the text widget. Note that the widget's area in the SageTV window (the entire bottom half of the screen) is outlined in yellow.

While the text widget is highlighted, press F2. You can change or even remove all the text in the widget's title. When done, press Return and then F6 to update the menu display. Note that the text in the SageTV window did not change – remember: the text widget's contents are overridden by its parent action widget.

Now, edit the title of the parent action widget (make sure that any text you enter is contained within the pair of quotes) and update the menu display. The display will change to match the text you entered.

Advanced Text Widget Properties

For extra credit, review the Text Widget Properties details and then experiment with the properties of either of the text widgets to see the results for any changes. To activate the properties editor dialog, highlight a widget, then press Ctrl+L.

Many properties changes will be seen in the SageTV window in "real time", as the changes are made, but a few changes may not become visible until you press Apply and/or use F6 while Studio is in focus to refresh the menu.

Tip: You can edit the properties for multiple widgets at the same time. Highlight both of the text widgets, then press Ctrl+L. Any properties that differ between the selected widgets will be grayed out; those that are the same for all selected widgets will show their values. Changing any of the values affects all selected widgets, while the values that differ will be left as-is.

Conclusion

This ends the **Text Display** tutorial, where you learned how to display text in the SageTV window. When done experimenting with any widgets in this menu, collapse the "Tutorial 02 – Text Display" menu widget tree.

Tutorial Set 3 – Shape Drawing

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 03 – Shapes". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various shapes drawn on the screen. **Reminder:** The use of F5 shifted focus to the SageTV window when the menu loaded, so be sure to switch back to Studio to continue using it. You can use F6 to refresh the current menu once it has been loaded.

Before continuing, you may wish to review the Shape Widget reference.

This tutorial really simply consists of noting the various types of basic shapes that can be drawn on the SageTV display. The "Tutorial 03 – Shapes" menu contains five shape widgets, each one drawing a differently styles shape. It is suggested that you experiment with the various shape widgets and their properties to see what happens. Among the details you may wish to note:

- ☮ Remember that, in the previous tutorial set, when a text widget was highlighted in Studio, its display area was outlined in yellow in the SageTV window. When a shape widget is highlighted, however, its display are is not outlined in yellow. Shapes are the only UI elements that are not outlined in this way.

- ☮ The name of the shape widget does not affect its display.

- ☮ Shapes are drawn in the order they are shown in the Studio tree, from top to bottom. If you change the color of a shape widget (via its properties) and make it overlap with another one, the shape widget lower in the tree will be on top of the one higher in the tree. A simple example: move the "Circle - filled" shape widget up one and refresh the display. (Highlight the widget, press Ctrl+U, then press F6.)

- ☮ The upper right square appears shaded. This is accomplished by adjusting the GradientAngle and GradientAmount property settings.

- ☮ Shapes are not drawn outside their allowable display area. They can be moved such that they overlap the edge of the allowable display area, leaving only a portion of the shape visible. Most of these shapes are children of the menu widget, but the large oval outline has been placed below a panel for experimentation purposes here. (Panels will be discussed later; for now just note that it exists.) If you experiment with position and/or size the "Oval - outlined" shape widget, you will find that there is more to the shape than what is shown on the screen.

Conclusion

This ends the **Shape Drawing** tutorial, where you learned how to display various filled or outlined shapes in the SageTV window. When done experimenting with any widgets in this menu, collapse the "Tutorial 03 – Shapes" menu widget tree.

Tutorial Set 4 – Image Display

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 04 – Images". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various shapes drawn on the screen.

Before continuing, you may wish to review the Image Widget reference.

Static Image Display

The two images on the left-hand side of the SageTV window are created by specifying the image to display in the properties dialog for those two image widgets. In Studio, look at the first two image widgets -- the ones without parent action widgets. If you highlight each one, you will see its display area outlined in yellow. Use Ctrl+L to view each widget's property dialog to see the filename for the image being displayed.

Dynamic Image Display

Now look at the other two image widgets in this menu – notice that each one has a parent action widget titled "SageIcon64.png". When an image widget has a parent action, the results of evaluating that action is sent to the image widget as the data for the image that is to be displayed. (In this example, a simple text string is used to pass the filename, but it could have been a more complex expression that dynamically built the name of the image to be displayed.)

Open the properties dialog for these two image widgets. Notice that both widgets actually list the filename of the image displayed in the Static Image Display example from above. Why is that not the image you see displayed in the SageTV window? The image determined from a parent action widget overrides the Image Source File property.

Pressed Image Display

Left-click on one of the two right-hand images in the SageTV window and hold the mouse button down. You will see the image change to match the one shown to its left. Open the properties dialog for the right-hand widgets. You will see that each one specifies a Pressed Image Source File that matches the Image Source File used by the left-hand image widgets. Note that the parent action widget does not override this image widget property.

Clicking on Images

When am image is clicked with a mouse, it can do one of two things:

☺ **Issue a SageTV Command** – The widget properties dialog contains a Fire User Event drop-box, where you can select an event to be issued. As an example, the bottom-left image (2nd from top image widget in Studio) will issue the Full Screen command, so if you click on that image in the SageTV window, it will cause the window to enter or exit full screen mode.

☺ **Execute Child Process Widget Chain** – When there is no Fire User Event setting in the image widget's properties dialog, clicking on an image will cause a child process widget chain to be executed, if there is one. If you click on the upper-left image in the SageTV window (top image widget in Studio), it will also toggle full screen mode, just like above, but it accomplishes that task via its child action widget. Note that the bottom-left image (2nd widget) has a child action also, but since it has a Fire User Event setting, that child action is not executed. (The child actions are very simple for purposes of these examples.)

Image Placement and Sizing

At this point, it would be a good idea to open the properties for the image widgets and experiment with the various settings for the image placement and sizing properties. Before moving the images around, notice that the top two images have the Preserve Aspect Ratio settings checked – this forces the images to retain their original appearance. In contrast, the bottom two widgets do not preserve their aspect ratios and have been resized to fit their entire allowable display area. Highlight each image widget to note its allowable display area in comparison to its actual display. Experiment with various settings for each image's Width and Height, and all the other settings to see how the image is affected. Most settings will take affect as you make the changes, but some may require selecting the Apply button and/or switching to Studio and pressing F6 to refresh the menu.

Conclusion

This ends the **Image Display** tutorial, where you learned how to display graphic images in the SageTV window and how they can cause events to occur when they are clicked with a mouse. When done experimenting with any widgets in this menu, collapse the "Tutorial 04 – Images" menu widget tree.

Tutorial Set 5 – Item Widgets (Buttons)

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 05 – Items (Buttons)". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the Item Widget reference. Essentially, item widgets are UI elements that can have focus and can be selected. In the SageTV window, these items usually appear as buttons; however, that button appearance is not their default way of being displayed. By default, the name of an item widget is used as the text displayed in the SageTV window for that UI element, and selecting the item performs no function.

Basic Item Display

In this tutorial's menu in Studio, look at the two item widgets titled "Thing 1" and "Thing 2". The names of those two item widgets are displayed directly below the menu's information text at the top of the screen. You will see that they do not look like selectable buttons, but if you use the arrow keys, focus will change from one item on the screen to another. You can also simply point to an item with the mouse to switch focus to it, if that mouse capability is turned on, or click on the item if focus does not automatically follow mouse movement. (That mouse option is selectable in the SageTV V4 Detailed Setup interface.) When focus changes from one item to another, the color of the text changes. That text color is controlled by the menu's theme widget, which will be covered in a later tutorial.

While one of the "Things" is highlighted in the SageTV window, press the Enter key , or left-click on a "Thing" with the mouse. Nothing happens because the item widgets do not have child Process Widget Chains.

Finally, notice that when one of the item widgets is highlighted in Studio, that item's display area is outlined in yellow in the SageTV window.

Setting Item Names

When an item widget has a child UI Widget Chain, the results of that UI chain are used as the displayed contents of the item instead of the name of the item widget. Look at the item widget named "Original Name". Notice that there is no such text displayed in the SageTV window. The displayed text for that item widget has been overridden by the child text widget titled "Renamed Item".

Note that in the first example, a simple single-widget child UI Widget Chain consisting of a single text widget has been used to show this principle; but, as described in the text

widget tutorial, action widget(s) could have been used to create the text to override the name of the text widget. The item widget named "Original Name 2" has a child action widget which overrides the name of the "Renamed Item 2" text widget.

These widgets do nothing if selected or clicked with the mouse.

Adding Functionality to Buttons

In Studio, look at the item widget titled "full-screen toggle". Highlight it and you will notice a yellow outline around the fairly simple-looking button called "Toggle Full-Screen Mode" in the SageTV window. Let's take a look at all of this item widget's children:

- ☺ **Button Background** and **Button Outline** – These are the basic shapes used to provide outline and background colors for the item, making it appear more like a button.

- ☺ **SageCommand("Full Screen")** – This single action widget is the entire Process Widget Chain that will be executed when this item is selected or clicked with the mouse – the SageTV window will enter or exit full screen mode if this button is selected. Note that since this action widget is part of a process chain, it is color-coded green.

- ☺ **"Toggle Full-Screen Mode"** – This is the action widget that sets the text to be displayed I the SageTV window for this button. Note that since this action widget is part of a UI widget chain, it is color-coded blue. It is **bolded** because it is the primary reference for this widget – if you right click on the widget and select Highlight References, you will see where else this widget is referenced.

The appearance of this 'button' is still fairly basic, since more advanced button displays would use widgets not yet covered, but it still functions as a basic selectable button.

Extra: Notice that the order of this item's child UI element widgets are quite important regarding its display in the SageTV window. To see the importance of widget order, move the "Button Background" image down one widget at a time and then press F6 to refresh the menu's display after every downward move to see the effect on the button's appearance.

Using Images on Buttons

In this menu, there is an item widget named "Picture Item" which has a child image widget ("Item's Picture") that is used as that item's display element. Normally, images do not receive focus, so they cannot be selected without using a mouse. By placing the image in an item widget, you can make the image selectable via the keyboard or remote.

In this example, there is no on-screen indicator to show that this item has focus. For now, simply note that this item has focus when no other text is colored red to indicate focus.

When this item is selected or clicked, the child "SageCommand("Full Screen")" action is executed. That child action is *italicized* because it is a secondary reference to that widget. Double click on it to jump to its primary reference.

Conclusion

This ends the **Item Widgets (Buttons)** tutorial, where you learned how to display and modify simple button items in the SageTV window and how they can cause events to occur when they are selected. When done experimenting with any widgets in this menu, collapse the "Tutorial 05 – Items (Buttons)" menu widget tree.

Tutorial Set 6 – Panel Widgets

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 06 – Panels". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the Panel Widget reference. A panel is basically used as a holder for other UI elements. Those other UI elements are placed in UI widget chains that are children of the panel, and their display is limited to the portion of the SageTV window used by the panel – a panel defines the clipping rectangle for drawing its children; in other words: a panel is the 'pen' where the child UI elements are contained. Most of the child widget properties for position and sizing are in relation to the size of the parent panel. (See Properties Common to Many Widgets.) Similarly, a panel is placed/sized in relation to its parent UI element, which could be the entire screen if its parent UI element is a menu widget. Panels may be children of other panels.

Unlike item widgets (buttons), panels do not normally gain focus. Scrollable panels are the exception to this rule.

Defining a UI Display Area

The first panel in the "Tutorial 06 – Panels" menu is named "Title Area". For this tutorial, the text description of the menu is contained as a child of this panel. Highlight the panel in Studio to see its area in the SageTV Window. Experiment with the panel widget's properties to see how the text display changes as the panel changes its size, position, etc. Remember: most settings will take affect as you make the changes, but some may require selecting the Apply button and/or switching to Studio and pressing F6 to refresh the menu.

Notice that as you reduce the Fixed Width property, the text display is repositioned to continue to fit within the panel. Also, because the text widget's Shrink To Fit property is checked, the text will be resized to fit the panel size.

When done, click on the Revert button for the panel's properties dialog and select OK.

The text widget's properties no longer sets its position in relation to the entire screen, as it did on previous tutorials when the description text was not a child of a panel; instead, it is placed relative to its parent panel. Also experiment with the text widget's properties to see how its position and size settings are relative to the panel, and its display is limited to the area within the panel.

AutoArrange Child UI Elements

How a panel's child UI elements are placed on the screen is determined by the panel's AutoArrange property. Depending on the setting, the child UI elements may be placed only at the position and size specified by the child, automatically placed with a size specified by the child, or automatically placed and sized.

Highlight the "AutoArrange Examples" panel. In the SageTV window, note that the yellow outline encompasses three lines of text, where each line of text is shown differently. Each line of text is contained in a panel with a different AutoArrange setting. Note that those lines of text actually consist of the same three text widgets, placed as references below the three AutoArrange panels. Their display differences are solely due to the panels' AutoArrange differences:

- **AutoArrange: None** – The panel does not automatically arrange its children at all. In this situation, the children are all responsible for their own placement and sizing. Because the text widgets have no size or placement properties set, they are all using the same default settings and end up overlapping each other.

- **AutoArrange: Horizontal** – The panel automatically places the children such that the next child begins where the previous child ends. Each child's width is determined by its own settings.

- **AutoArrange: HorizontalGrid** – The panel divides the display area into equivalently-sized areas and places the children into those grid locations.

Note: these are not all the possible AutoArrange settings for various widget types, but they should show results that can be applied to the other settings.

To see how these AutoArrange settings interact with the child widget settings, highlight one of the text widgets below any of the AutoArrange panels. Since each line references the same text widgets, highlighting any of them will highlight that widget on all three lines. Experiment with changing the widget's properties to see how the changes affect the same widget on all three lines.

Navigation Between Panels

Normally, navigation between focusable UI elements is handled by checking for the next focusable element in the direction of the arrow pressed (Left, Right, Up, Down). So, when buttons are on the screen and an arrow is pressed, SageTV looks for the next button in that direction and sets focus to that button. This change in focus can extend to focusable UI elements outside the panel where the currently focused element resides. However, it is possible to override this such that focus stays within the current panel, wrapping focus back to another element within that panel. To see this in action, refer to the widgets below the panel titled "Navigation Examples". On the SageTV screen, those

widgets result in two columns of very basic focusable items, with A, B, C, and D in the left column, and 1, 2, 3, 4 in the right column.

With the mouse, make sure that "1" is highlighted in the SageTV window. Use the Up or Down arrows repeatedly to rotate focus through all the items in that column. Notice how focus wraps from the top to bottom, and vice versa. These items are contained in the "Upper Right" and "Lower Right" panels in Studio. As you can see, focus can change from an item in one panel to an item in the other.

Now, use the Left arrow to change focus to the left column. Use the Up and Down arrows repeatedly again. Notice how focus stays within either the top two items or the bottom two items, depending on which group has focus. This is because the "Upper Left" and "Lower Left" panels have the Wrap Vertical Navigation property checked. When that property is checked, focus will wrap within that panel in the vertical direction when the last item is reached, instead of shifting focus to an item in another panel.

Conclusion

This ends the **Panel Widgets** tutorial, where you learned how to use panel widgets to create sections in the SageTV window to limit where the panels' child UI elements can be drawn, how to arrange UI elements within those panels, and how panels can affect focus navigation. When done experimenting with any widgets in this menu, collapse the "Tutorial 06 – Panels" menu widget tree.

Tutorial Set 7 – Action Widgets

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 07 – Action Widgets". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the Action Widget and Widget Chain Types references. In short: action widgets contains lines of code that SageTV is to execute while preparing to build the data to be shown in the SageTV window (**UI widget chain**), or in response to a program or user input event (**process widget chain**). UI widget chains get executed every time SageTV needs to build the UI elements to be displayed. Process widget chains only get executed when some event occurs to cause the code to be executed; that event could be in response to user input (select a button on the screen or pres a button on the remote) or as a result of an event internal to SageTV (such as a Hook Widget).

Widget Chain Types

To see the various types of widget chains, take a look at the widget tree below the "REM Widget Chain Types" action widget. The widgets in this tree do nothing useful, but they do serve to show the various widget chain types.

- ⚙ **UI widget chains** (blue indicator) – Notice that all the action widgets that lead directly to UI element widgets (the item and text widgets) have their icons color coded blue. This means that those widgets will be executed as SageTV determines how to display those UI elements. Only widgets that lead directly to a UI element are executed during this process.

- ⚙ **Process widget chains** (green indicator) – Below the item widget, there is a child widget tree where the actions are color coded green. Such actions are only executed when the parent item (button) is selected. Note that these actions do not have to be in a continuous parent-child linked tree; as you can see, the first green coded action has two child branches.

- ⚙ **Not executed** (yellow indicator) – Yellow coded widgets are not executed. Such widgets usually are seen in a UI widget chain, where some of the widgets do not lead to a UI element widget. That is the case in the example seen here: some action widgets are in a branch of a UI widget chain that does not lead to any type of UI widget. **Note:** This is a common error when creating data to be displayed in SageTV. If some data manipulations are not reflected in the SageTV window, check for any yellow coded widgets in that section of code.

Tips: There is currently no widget type for comments. One easy way to set aside widgets to be used as comment lines is to use an action widget and place the comment inside quotes, making it a string. Prefixing the string with something like REM (for 'Remark') makes it easy to see which lines are intended as comments.

You may wish to use 'comment' actions as the widget to be used as the reference widget for reusable sections of code. That way, the comment line could describe what that referenced code will accomplish. And, if the referenced code expects any variables to be set with certain values, you could list those variable names and how they should be set.

Widget Chain Execution Order

Note: This tutorial section uses a variable named "DisplayString". Variables will be discussed in more details in the next tutorial set.

The third line in the SageTV window when the "Tutorial 07 – Action Widgets" menu is active contains "Testing … 1 2 3 4 5 6 7 8". That text string is displayed below the "REM Display execution order results" action, but the string is created under the "BeforeMenuLoad" hook. (Note that the actions below that hook are color coded green, so they are part of a process widget chain.) The widget chain below the hook contains multiple branches; each line that adds more text to the "DisplayString" variable is numbered according to the actual execution order, as you can see by the order the numbers are displayed in the SageTV window.

You should experiment with the order of those actions by moving some of them Up or Down. Don't forget to use F6 to refresh the menu after each modification. As you change the widget order and refresh the SageTV window, you will see the numbers change to match the new execution order.

Extra: In addition to simple moving widgets up or down, try moving some of the widgets or creating secondary references by dragging one of the "DisplayString = DisplayString + "n "" action widgets to another of those action widgets. **Caution:** Be careful not to create a reference to a parent widget below one of its children (do not reference #5 below #7, for example), as that would create an infinite loop. Loops will be covered in a later tutorial. A safe secondary reference would be to drag #2 to #8.

SageTV API Function Calls

Action widgets can contain calls to any of the SageTV API functions. Examine the widgets below the "REM API Calls" action widget. The first action contains a call to the API function for "Max(Value1,Value2)". Look at that widget's properties to see how the API call is listed, along with the required list of parameters. The results of that function call are added to the string listed in the Prefix filed of the properties dialog, and then displayed in the SageTV window via a text widget.

While the action widget properties dialog is limited to displaying a single API call and its parameters, actions may actually contain multiple API calls. The second action widget contains calls to the same "Max" function, but also calls "Min" with the same parameters. Look at the properties of that widget to see how it compares to the properties of the first action hat only called "Max". Notice that the second API call is simply listed as part of the Suffix property field.

Automatic Type Conversion

While executing code on action widget lines, SageTV will automatically convert data types as needed, when possible. This was already seen earlier where some of the numbers added to the DisplayString variable were text strings and some were numbers, and in the examples for calling the Max and Min API functions. In both of these situations, numbers were automatically converted to text as the text strings were built.

As mentioned in General Expression Information, some SageTV internal data types will be automatically converted also, when there is a 1-to-1 correspondence between the two variable types. As mentioned in that section, for example, the MediaFile and Airing types can be automatically converted since they have that 1-to-1 correspondence.

Tip: When using number values returned from GetProperty(), it may be useful to multiple the function return value by 1 or 1.0 to force the result to be a number. Look at the widgets below the "REM Data Type Conversions" action widget. Those widgets create the last line displayed in the SageTV window, which consists of three different numbers:

1) **GetProperty()** – The result of this API call is simply displayed.

2) **GetProperty() + 100** – The result of this call has **100** added to it. Notice that the result of GetProperty was actually a string, so 100 was converted to text and added to the string. In the SageTV window, this resulted in "100" being displayed at the end of the GetProperty results shown in #1, above.

3) **GetProperty()*1 + 100** – The result of the GetProperty call was multiplied by 1 to convert it to a number, then 100 was added. In the SageTV window, this resulted in the display of a number that is 100 greater than the GetProperty results shown in #1, above.

Conclusion

This ends the **Action Widgets** tutorial, where you learned how action widgets can be part of process or UI widget chains, in what order action widgets are executed, how to call SageTV API functions, and the fact that some data types can automatically be converted from one type to another. When done experimenting with any widgets in this menu, collapse the "Tutorial 07 – Action Widgets" menu widget tree.

Tutorial Set 8 – Variable Usage

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 08 – Variables". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the Attribute Widget and 4) Attributes / Variables references. Like most programming languages, the Studio language uses variables to track various pieces of data so that the information is available for use or manipulation later. The attribute widget is used to declare and initialize a variable; however, if a variable is not declared by using an attribute widget, it can be automatically declared when it is first used to the left of an equal sign.

Using the "this" Variable

SageTV stores the results of any expression in the "this" variable, which can be accessed by the next expression. Of course, the current contents of "this" will be overwritten by the results of the next expression, and the next expression, and the next, and so on. To see an example of this, look at the widgets below the action widget titled "REM using 'this'". Here is what happens:

- ☯ The first child action widget adds 5 and 7, which gets stored in "this".
- ☯ The next action widget creates a string to show the value of the "this" variable. The resultant string is stored in "this", replacing the sum of 5 and 7.
- ☯ The text widget uses the contents of the "this" variable as the text to be displayed; that text can be seen in the SageTV window, directly below the menu's title line.

Declaring and Accessing Variables

As mentioned previously, variables can be declared via an attribute widget or they may be created automatically when they are first used. Simple examples of this can be seen under the panel titled "Declaring and using variables". Highlight the panel in Studio to see it outlined in yellow in the SageTV window. That panel contains code that displays a few lines of text and a button in the SageTV window; within that yellow outline you will see:

- ☯ The 1st line shows the value of the "DefinedVariable". This variable was declared and initialized in an attribute widget directly below the panel. To see the value an attribute widget is initialized to, just view its properties. **Note:** the attribute widget did not have to be the first child of the panel; in fact, if you move it down a few positions and then refresh the menu, you will see the same results.

☼ The 2nd line shows that the variable declared in the attribute, "DefinedVariable", is still accessible below a panel. The new variable context created by the panel does not interrupt the attribute-declared variable.

☼ The 3rd line displays the results of automatically creating and setting the "AutoVariable" variable.

☼ The 4th line shows that the automatically declared variable, "AutoVariable", is NOT accessible below a panel. The new variable context created by the panel (or other UI elements) does not retain automatically-declared variables. **Note:** As shown by the example on line 2, if you wish to access local variables below panels in later widgets, declare that variable via an attribute widget.

☼ The 5th line accesses a variable called "NoSuchVariable", which has not been declared or initialized previously. Since this the first time it has been used, SageTV will automatically declare the variable and set its value to "null", which is the value you will see displayed in the SageTV window.

☼ The 6th line is actually a very basic button. A variable named "ButtonVariable" is declared below the button's item widget and its value is shown as part of the text on the button.

☼ The 7th line of text attempts to access the button's "ButtonVariable" variable, but as you can see, its value is shown as "null". Why? Remember: variables exist within a specific context, which is the tree extending as child widgets below the point at which the variable is declared. Variables do not exist outside the scope of their context. (See Variable Context (Scope).)

Accessing Out-of-Scope Variables

The easiest way to access a variable that isn't within the context of the widget where you wish to access it is often to simply declare the variable in an attribute widget at a point in the tree that is a parent of all places where you wish to access that variable. At times, however, this is not possible. But, there is a way to gain access to variables that exist in the context of the UI element that currently has focus. Examples will now be shown to access the variables decalred via attribute widgets under the "Declaring and using variables" panel and the "Button with a child attribute" item widget.

Highlight the "Variable not in scope" panel in Studio to see it outlined in yellow in the SageTV window. That panel contains code that displays a few lines of text; within that yellow outline you will see a header line followed by:

☼ Line A attempts to access the "ButtonVariable" and "DefinedVariable" variables directly, but as you can see, they are both displayed in the SageTV window as "null", showing that they do not exist in the current context.

⊕ Line B uses the API function "GetFocusContext()" to gain access to the context of the item that currently has focus. As you can see in the SageTV window, the "ButtonVariable" and "DefinedVariable" variables are now accessible, since their values are displayed.

⊕ Line C has been placed under a panel, which is under the call to the "GetFocusContext()"API function. The "ButtonVariable" and "DefinedVariable" variables are no longer accessible again. Why? The results of the call to "GetFocusContext()" reside in a temporary context area that is not retained when a new context is created for the new panel.

Conclusion

This ends the **Variable Usage** tutorial, where you learned the basics of declaring and accessing variables. When done experimenting with any widgets in this menu, collapse the "Tutorial 08 – Variables" menu widget tree.

Tutorial Set 9 – Conditionals and Branches

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 09 – Conditionals and Branches". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the Conditional Widget and Branch Widget references. The conditional widget is used to test the result of an expression and then to either 1) execute its child widget chain if the result is true, or 2) execute one or more branches whose evaluation results match the conditional's results. The conditionally executed widget chains could be either UI or Process chains.

Basic Conditional Statement

The most basic usage of the conditional widget is to use it with an expression that could evaluate to 'true', and then to add a child widget chain (with no branch widget) that you wish to have executed only when that expression is true. In this tutorial's menu, look at the widget chain below the "REM Basic Conditional" action widget. The first conditional tests the value of "TestVar" and has a text widget as its direct child; notice that this conditional has no child branch widget. Since TestVar is currently set to true, that text widget's contents are displayed in the SageTV window.

Now look at the second conditional widget, which also tests the value of "TestVar". It has a branch widget as its child, and that branch simply contains 'true'. This conditional + branch(true) pair of widgets results in the same functionality as just using a conditional widget: the child widget chain is executed ONLY if the conditional's expression evaluates to 'true'.

Edit the "TestVar = true" action widget so that the variable is set to something other than true; perhaps use something like "TestVar = trueX". After refreshing the menu display, you will see that the two lines of text are no longer displayed since the conditional no longer evaluates to 'true'.

Note: Notice that the conditional and branch widgets in this example are color coded blue, indicating they are part of a UI widget chain. Like action widgets, conditionals and branches are color coded according to widget chain type.

Conditional + Branch(es) Usage

When using a conditional widget plus one or more child branch widgets, the conditions tested can be more than simple checks for evaluating to 'true'. Pretty much any expression can be used on the conditional and all of the child branches. After all of those expressions are evaluated, all branches whose results match the results of the

conditional's results will have their child widget chains executed. This means that it is possible for multiple branches to be executed; while that may not be something that is purposely done often, it is possible.

Remember: When SageTV is checking multiple branches underneath a conditional, it will check each branch one at a time, and if that branch's value matches the conditional then the action chain underneath that branch will continue to be evaluated. SageTV does not evaluate all of the branch expressions before deciding what to do; therefore, do not use expressions which may result in changes that affect how the next branch evaluates.

Tip: An easy way to check which of several conditions is true is to set the conditional widget to 'true' and then set each branch widget to one of the expressions to be evaluated. If all those branches are mutually exclusive, then only one will be executed. This is done in the tutorial menu, as described below.

The 'else' branch: If no branch results match the conditional's results, then SageTV checks for a branch titled 'else'. If that branch exists, its child widget chain is executed. If it does not exist, then no branch is executed. Note that if the 'else' branch executes, by definition, no other branch will execute along with it.

IMPORTANT: A conditional widget should not have both a direct child widget chain AND child branches. This syntax is not supported and has undefined results.

For an example of using a conditional widget with multiple branches, see the widget chain below the "REM Conditional with Branches" action widget in this menu. That widget chain has two conditional widgets: one for "If CurrentValue" and one for "If true". The buttons for the first If statement are shown in the upper box in the SageTV window, while the buttons for the second If statement are shown in the lower box. In both boxes, only certain buttons are displayed, depending on the value of "CurrentValue" and the branch widget that is the parent of each button. Selecting one of the active buttons modifies the "CurrentValue" variable as shown on the button, then refreshes the menu via a call to the "Refresh()" API function.

Look at the first conditional ("If CurrentValue") and the branches below it. Note that the first branch, "true", is never executed because the "CurrentValue" variable is never set to 'true', the "else" branch is only executed when no other branch executes, and the other three branches are executed only when the variable matches the values shown in Studio. For this conditional and its branches, only a single branch is ever active at one time; therefore, only one of the four buttons are shown at any time.

Now look at the second conditions: "If true". As mentioned in the tip, above, this conditional and its branches are set up to handle whatever branch expressions evaluate to true, instead of using an expression on the conditional with possible results on each branch. Also note that because of the contents of each branch expression, it is possible for multiple branches to be executed; in fact, regardless of the value of the "CurrentValue"

variable, the last branch will always be executed because its expression is simply the value "true". So, in this box there will be multiple active buttons.

Finally, note that the branch containing "CurrentValue > 10" is color coded yellow. That is because the branch is part of a UI widget chain, yet it does not lead to any type of UI widget element. Such parts of a UI widget chain are ignored and never executed, thus the yellow widget chain type indicator.

Conditionals in Process Widget Chains

So far, all of the conditional widgets discussed in this menu have been part of a UI widget chain, but they work the same way when they are part of a process widget chain. The branch for the bottom button on the screen, "If true", contains a conditional as part of the process widget chain that is executed when the button is selected. It simply modifies the "CurrentValue" variable a different way, depending on its current value.

Conclusion

This ends the **Conditionals and Branches** tutorial, where you learned the basics of using conditional and branch widgets to conditionally execute code or show UI elements. When done experimenting with any widgets in this menu, collapse the "Tutorial 09 – Conditionals and Branches" menu widget tree.

Tutorial Set 10 – Loops

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 10 – Loops". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Creating a Basic Loop

At its most basic level, creating a loop in Studio simply involves adding a widget reference to another widget higher in the same widget chain AND adding some way for the loop to exit. Simply creating a widget chain like **Action 1** → Action 2 → Action 3 → *Action 1* will create an infinite loop, since there is no way to exit that widget chain. At some point in that widget chain, there should be a conditional widget that will eventually evaluate to a condition that stops the loop.

Look at the pseudo-code widget chain below the "REM Basic loop construct" action widget. The first thing to note is that this widget chain is color coded yellow, so you can tell that it will not be executed. The second thing to notice is that it doesn't contain valid code anyway; it is just a sequence of text strings (widget titles in quotes) showing one basic way to create a loop:

1. Initialize all the data to be used in the loop.
2. Test the loop condition(s) to see if the loop should continue. Should it continue?
   a. **YES** – Manipulate the data, including any data used to determine whether to continue the loop, then add a reference back to the widget at the start of the loop.
   b. **NO** – For a UI widget chain, continue the rest of the UI chain. For a Process widget chain, either continue the rest of the chain, or end the current one.

Of course, this is not the only widget sequence that can be used in a loop; it is just used as an example. Like in most other programming languages, you can control where you increment your loop control variables and whether to do the loop condition testing before or after the first iteration of the loop. It all depends on where you place your reference widget back to the start of the loop and where you place the loop condition test.

A Sample Loop

To see a sample basic loop in action, look at the widget chain below the "REM Basic loop" action widget. This loop uses the basic construct described above to add the number 1 through 10, then display the result on the second line in the SageTV window.

Preventing Infinite Loops

If SageTV gets caught in an infinite loop because the loop condition never allows the loop to exit, there is no way to break into the code to stop execution. If an infinite loop does occur, you will have to stop the SageTV process, restart SageTV/Studio and edit the code in the loop before it executes again. While loops like the sample shown above are fairly simple in terms of its loop condition, there are times where the loop control variables and/or the loop condition are much more complex. In these situations, it may be a good idea to use a 'safe' loop construct that can prevent infinite loops – essentially a loop limiting technique can be used.

The widget chain below the "REM Basic Safe loop" action widget shows such a loop. Just like the previous sample loop, this loop is designed to add the numbers 1 through 10. But, in this case, the loop control variable, "LoopIndex", is never incremented in order to simulate a faulty loop. A loop limiting control variable, "MaxLoopCount", has been added to the loop to make sure the loop executes no more than a specified number of times, in this case: 50. If this limiter had not been added to this loop, it would be an infinite loop.

While the loop limiting variable also has to be implemented correctly in order to prevent an infinite loop, remember that the sample loop control variable, "LoopIndex", is a fairly simple way to control a loop. In a real-life loop where the loop condition may be controlled by the return value of a function call, or by a list of data, it may be possible to run into unexpected situations where the data prevents loop conclusion. Using a loop limiter in such a situation is one way to prevent SageTV/Studio from no longer responding. Once the code development and debugging phase is complete, the loop limiter can be removed or left in place as a safety measure.

UI Elements in Loops

Simply put: UI widgets may not be used in loops to create nested UI elements, with a single exception: shapes.

Look at the widget chain starting at the "REM Loop with shapes" action widget. The loop itself is very similar to the previous sample loops, except that it draws 10 shapes inside a panel widget rather than adding numbers. (A simple shapes-within-a-loop method is used in the default SageTV STV when drawing the green segments on the playback OSD to indicate which portions of a show have been recorded when the recording is made up of multiple parts.) Adding other UI widgets to the loop with the shape widget, or replacing the shape widget, will not cause those other UI elements to be shown repeatedly, if they are shown at all.

Restoring "Hidden" Loop References

While looking at the "REM Loop with shapes" widget chain, notice that it draws a semi-transparent shape, "Color Fill", as the background color for the panel. What happens if

we wish to make it the semi-transparent foreground color instead? Highlight the bolded "**MaxLoopCount = MaxLoopCount - 1**" action widget and use Ctrl+U to move that widget above the "Color Fill" shape widget. Did you notice any interesting side effect of that widget move? The loop where the shapes are drawn has completely disappeared!

When you move a primary reference, it becomes a secondary reference and shifts the primary reference to one of the widgets that used to be a secondary reference. The way to find the new primary reference is to double click on the *italicized* secondary reference. So, do that. But, when you double click on the italicized *"MaxLoopCount = MaxLoopCount - 1"* action widget, Studio does not jump to the primary reference. The loop is just gone and is nowhere to be found. However, if you refresh the menu, the shapes inside the loop are still drawn – they are now behind the semi-transparent "Color Fill" shape, as you would expect based on the widget order. This must mean that the loop still exists… <u>somewhere</u>! Remember: Studio is just displaying widget chains in a tree format and *italicized* secondary references do not have their child widget chains displayed. They still link to widgets, however. In this case, the only other reference that could have been set as the new primary reference after the widget move is a child widget of the one that was moved, so it is not displayed anywhere in the tree.

The solution: If the entire loop seems to disappear after moving some widgets around, simply set the *italicized* widget as the primary reference. In this case, right-click on the *"MaxLoopCount = MaxLoopCount - 1"* action widget, then select 'Set as Primary Reference'. You may need to expand the tree after doing that, but the entire loop has been restored to visibility.

Conclusion

This ends the **Loops** tutorial, where you learned the basics of using widget references and conditional widgets to create loops. When done experimenting with any widgets in this menu, collapse the "Tutorial 10 – Loops" menu widget tree.

Tutorial Set 11 – Pop-up Options Menus

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 11 – Options Menus". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and various items drawn on the screen.

Before continuing, you may wish to review the OptionsMenu Widget reference. The OptionsMenu widget is used to open a new interface dialog on top of the existing SageTV display.

Creating and Closing Options Menu Dialogs

An options menu is created only as a part of a Process widget chain below such widgets as a button, hook, image, or listener. It is not created while a UI widget chain is executing, so it could not be a direct child of a Menu widget, for example. In other words, an options menu is created in response to some event, whether it is user input or an event that happens in the SageTV core.

Near the top of this menu is an OptionsMenu widget titled "Unseen Options Menu". Since this OptionsMenu is a child of the Menu widget, it will not be executed and displayed on the SageTV window.

When an options menu is created and displayed, its UI child widget chains are shown within the allowable borders for that OptionsMenu widget, similar to the display of a the UI elements under a Menu widget, except that a Menu widget encompasses the entire screen while an OptionsMenu widget could cover all or only a portion of the screen.

Look at the widget chain below the panel titled "Displaying Options Menus". This panel contains a single button that is displayed in the SageTV window: "Click me to pop up an Options Menu". If you select this button, a basic pop-up options menu will be displayed via the "Sample Options Menu" widget below the "Options Menu 1 button". Notice that the dialog does not cover the entire SageTV window, so you can still see the rest of the display, but you cannot interact with any part of the display except for the options menu. While an options menu is active, it receives all input events. You can try clicking on the buttons visible in the background, but they will not do anything. Therefore, OptionsMenus in SageTV are like modal dialogs.

In the pop-up dialog, you will see two button choices. The first one simply closes the current dialog by calling the "CloseOptionsMenu()" API function. The second button opens another dialog on top of the first dialog, with the same situation as before: you can only interact with the top-most options menu.

The second pop-up dialog also has two buttons. The first one simply closes the current options menu, while the second one closes both options menus via two calls to the "CloseOptionsMenu()" API function. **Note:** each call to "CloseOptionsMenu()" closes only the most recently created options menu. Don't forget to call that function once for each options menu that you create.

Widget Chain Execution After Closing an Options Menu

When using options menus, you should be aware of one very important side effect that an options menu has on the continuing execution of the Process widget chain that it is a part of: when an options menu closes, execution will only continue with any widget chain that is a child of the OptionsMenu widget's parent widget. The next section of the SageTV uses the widget chain below the "Options Menu closing effects" panel to show this effect.

Select the "Execute widget chain" button in the SageTV window. Notice how the text displayed after "Last setting:" changes – it will contain all the numbers from 1 through 9. This text is created by the Process widget chain below the button titled "Normal widget chain execution" in Studio. That chain contains several action widgets to create the string, along with a single call to "Refresh()" to update the display. Every one of those actions gets executed.

Now, select the next button, "Use Options Menu in chain", in the SageTV window. An options menu will appear, asking you to click on the "Continue" button. Do that. Notice that the "Last setting:" display changes, now including only the numbers 1 through 5. In Studio, look at the Process widget chain below the "Options Menu widget chain execution" button. That widget chain is the same as the one below the other button, except that an OptionsMenu widget was inserted below line 4 and extra calls to "Refresh()" were added in order to make sure the SageTV window gets updated after the last action widget executes. From the results shown in the SageTV window, you can see that no other text was added to the "LastText" variable after adding the number 5. As explained at the start of this section, widget execution did not continue further up the widget chain after the options menu was closed. (If there had been additional widgets below the one where 5 was added to the string, they would have been executed. You could try adding some to see that happen.)

Because of the effect seen here, it is not advisable to include pop-up options menus in a widget chain like the one shown here. Once you add the OptionsMenu widget to a Process widget chain, be sure that all further widget chains are children of the OptionsMenu's parent widget.

Conclusion

This ends the **Pop-up Options Menus** tutorial, where you learned the basics of using OptionsMenu widgets to display pop-up dialogs – how to create them, close them, and how they affect the continuing execution of the current widget chain. When done

experimenting with any widgets in this menu, collapse the "Tutorial 11 – Options Menus" menu widget tree.

Tutorial Set 12 – Tables

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 12 – Tables". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and a couple buttons for selecting which table to view.

Before continuing, you may wish to review the Table Widget and TableComponent Widget references. From those references, you will see that both of those widgets types must be used together in order to create scrollable lists of items, or tables, in SageTV. SageTV can create tables with a vertical OR horizontal dimension, a one dimensional table, or with both vertical AND horizontal dimensions, a two dimensional table.

One Dimensional Tables

Select the "Show a 1-D Table" button to see an example 1-D table displayed in the SageTV window. Such tables are used in many places in the default SageTV STV, usually whenever a list of selectable items is shown on the screen. To see the Studio code for this table, look at the "1-D" branch below the "Table Area" panel.

Notice how this table is constructed:

- The list of available recording qualities is retrieved via the "GetRecordingQualities()" API call. For a one dimensional table, the data for the table's list is 'fed' to the Table widget via its parent Action widget. In this case, the "GetRecordingQualities()" call returns a list of all the available recording qualities, which then get sent to the Table in the following widget.
- The "QualityListTable" Table widget sets up the table's display area, dimension, and other settings.
- The "CurQuality" TableComponent widget is used as a **Cell** that represents each element in the list. The name of this widget is the local variable used to access the current list element. While there is only a single cell TableComponent widget, that one widget is used to represent every list element, and its child UI widget chain determines how that table/list element gets displayed on the screen.
- Below the "CurQuality" cell widget, an Item widget was used to display the each cell as a button. **Note:** The table elements to not have to be shown as buttons; they could have been displayed as non-selectable simple text, or anything else you want them to appear as.
- Below the Item widget is a widget chain that determines the appearance and behavior of the button. (Selecting a recording quality button shows some info about that setting in a pop-up dialog.)
- **Extra:** Also below the "QualityListTable" Table widget, there is a panel titled "VPagination", where the page up/down icons are displayed as needed, using the "IsFirstPage" and "IsLastPage" automatic local variables. (See Predefined Local

SageTV Studio
Page 104 of 138

Variables.) In addition, the "TableRow" was used to show the list number on each button.

Things to do: Try highlighting some of the widgets below the "1-D" Branch widget to view their yellow-outlined display areas on the screen and how they relate to each other. You may also want to try changing various widget property settings to see what effect each change has. Remember: while many settings will show their changes in the SageTV window in real-time as the changes are made, some other settings may require that the menu be refreshed (press F6 in Studio) or to be reloaded (highlight the Menu widget and press F5). You may wish to experiment with the "QualityListTable" Table widget to change its NumRows and/or NumCols settings.

Two Dimensional Tables

Select the "Show a 2-D Table" button to see an example 2-D table displayed in the SageTV window. To see the Studio code for this table, look at the "2-D" branch below the "Table Area" panel. **Note:** This two dimensional table uses buttons in what you might consider to be non-standard places in a table. This was done on purpose to show that there is no single place in a table where buttons have to be used.

This two dimensional table is constructed quite a bit differently from the one dimensional table shown previously:

- ☯ The first widget for this table is the "QualityList2DTable" Table widget, not an action to retrieve any list of data. As before, this widget sets the table's display area and dimensions. Unlike one dimensional tables, 2D tables do not have the table data fed directly to the Table widget; instead, the table's contents are fed to the TableComponent widgets in the child widget chains. Below this Table widget can be found four additional widget chains:
    1) The widget chain starting at the "Recording Quality Title Panel" panel is just an optional item placed to use the area not filled by the row header, column header, or table body. Such an open, unused space is dependent on how the two dimensional table is laid out. The LiveTV Guide grid in the default SageTV STV, for example, uses this leftover space differently.
    2) The column header widget chain uses a **ColHeader** TableComponent type to list the titles for the two columns used for this table: the Format and GB/hr columns. The "QualityInfo" title of the TableComponent widget is used later in the table as the column for the current **Cell**. These elements are displayed as text in the SageTV window. In this example, the column headers receive their data from the "DataUnion()" API call right before the TableComponent widget.
    3) The row header widget chain uses a **RowHeader** TableComponent type to list the titles for the rows of recording qualities used for this table, retrieved from the "GetRecordingQualities()" API call which has been fed to the TableComponent widget. The "CurQuality" title of the TableComponent widget is used later in the table as the row for the current

Cell. Note that most of the items in the row list are displayed as selectable buttons, but for a change of pace, one list element is shown only as a non-selectable text entry. (As in the one dimensional table, select a button to show some info about that setting in a pop-up dialog.)

4) Finally, the "QualitySettings" **Cell** TableComponent is used to fill in the body of the two dimensional table. Note that in this example, no data has been fed to the TableComponent; instead, it retrieves the data to be displayed via the row and column TableComponent variables: "QualityInfo" and "CurQuality". The "QualityInfo" variable (see #2, above) is used to determine which column a cell belongs to; those in the first column are displayed as buttons, while those in the second column are displayed as text. Then, the "CurQuality" variable is used to display the info about the recording quality; that variable was set in #3, above, according to which row the cell belongs to. (Selecting a button does nothing. As an exercise, you could add a Process widget chain to do something when a button is selected.)

The visual layout of this two dimensional table is:

Simple Text to fill left-over space	Column Header, from **ColHeader** TableComponent.
Row Header, from the **RowHeader** TableComponent.	Table body, created by the **Cell** TableComponent type. Uses the row header and column header variable names to access and display the table contents.

The table could have been displayed differently, depending on how you wanted a table to be presented to the user, but the above layout was chosen for this example table. Each element of the table was placed and sized for its desired location. You can experiment with the widget properties to see how changes affect the table and its layout.

As mentioned previously, various elements of the table were chosen to be buttons or text simply to show that there is no rule that any certain table element has to be a selectable

button – they just have to be UI elements that get shown in the SageTV window. It is up to the developer to determine which UI elements are to be selectable.

Additional Table Scraps

Before ending this tutorial set, there are a couple advanced tips regarding tables that you might find useful:

Time spans in tables Most of the time, the table cells are all set to the same width and height, determined by the number of rows and columns to be displayed and the amount of screen space to be used for the table. However, when a column or row is fed the results of a call to the "CreateTimeSpan()" API function, SageTV will be able to automatically size the cell to match the timespan for a cell's contents when the cell contains a database item that has a time span. An example of this can be seen in the code that creates the LiveTV Guide's grid for the default SageTV STV. For that table, the **ColHeader** TableComponent uses the results of a call to "CreateTimeSpan()". Since the table's cells contain Airings, and Airings have a time span consisting of start and end times, SageTV is able to automatically adjust each cell's width to match its time span.

Dynamic table size adjustments The size of a table (the number of columns and/or rows it displays) is set in the Table widget via the NumRows and NumCols settings. These values are only referenced when SageTV is initially setting up a menu and determining the layout of all of that menu's UI elements, so if you try to set them to dynamic values (see General Widget Properties) and then calling Refresh() to redisplay the menu when those variables are changed, the table will not resize itself to match its new dimensions. One way around this is to call a reference to the same Menu widget after using the "AddStaticContext()" API function (see Variable Context (Scope) for more comments on that call) to send values for the table's new size to a new instance of the same menu. This technique is used in a few menus for the default SageTV STV – see the "BeforeMenuLoad" hook and associated variables for the LiveTV Guide, Video Library, and Picture Library menus.

Conclusion

This ends the **Tables** tutorial, where you learned the basics of creating and displaying one and two dimensional tables. When done experimenting with any widgets in this menu, collapse the "Tutorial 12 – Tables" menu widget tree.

Tutorial Set 13 – Listeners

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 13 – Listeners". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and a line saying what the last listener action was.

Before continuing, you may wish to review the Listener Widget reference, which explains how listeners are activated by user input and that when activated, they execute their child Process widget chain. Child UI widgets are not displayed in the SageTV window, however a listener could be used to display a pop-up options menu.

Listeners can be children of most UI widgets (see Valid Widget Parent-Child Relationships), but this tutorial has placed all but one of the sample listeners below the Menu widget for simplicity.

Basic Listener

In this menu, a basic Play command listener is used – in this case, it simply sets the "LastSelection" variable to indicate which listener was called and then refreshes the section of the screen where the last selection information is displayed, via a call to RefreshArea("Info Panel"). Enter the Play command (Ctrl+D) to see the screen update after the listener is activated.

Obviously, in a 'real-life' STV, you could have created a more useful Process widget chain below this listener to do something more useful when the Play command is received.

Mouse Listeners for Non-Focusable UI Elements

Unlike other listeners, mouse listeners can be activated for UI elements that can not receive focus when using a keyboard or mouse. An example of this is the mouse listener below the Text widget used to display this menu's title line. Highlight the title line's Text widget to see its display area outlined in yellow in the SageTV window. If you click anywhere within that yellow rectangle, the mouse click will be reflected on the info line, in response to the widget chain that is executed below the mouse listener.

The title line's Text widget has another listener for the Left command. However, since the title line can not receive focus, that listener will not be activated. While SageTV has focus, use the Left command and notice that no change is made to the info line.

Next, highlight the "Info Panel" widget in Studio to see its yellow outline in SageTV. If you click the mouse below this rectangle, the mouse listener that is a direct child of the Menu widget will be activated and the info line will be updated to reflect that. But, if you

click inside the yellow outline for the "Info Panel", no info line update occurs. Why is a mouse click reflected when clicking on the title line or on the bottom portion of the screen, but not when the "Info Panel" is clicked? The mouse listener is being activated for the UI element that you click on. The "Info Panel" widget has no child mouse listener, so a mouse click on it does nothing. The bottom portion of the screen is not covered by any other UI element, so the mouse listener under the Menu widget gets activated.

Listener-created Local Variables

Some listeners can set local variables in accordance with the data input received. (See Listeners That Set Local Variables.) In this menu, look at the "NumberEntry" listener and its widget chain. First, look at the properties for that widget. Note that it is actually a "Numbers" listener with a title of "NumberEntry", thus "NumberEntry" becomes the local variable containing the user-entered number. When you press a number, SageTV will update the info line to indicate the number that was entered. **Note:** If this listener had not been named, the number entered could be accessed via the '**this**' variable.

Adding Command Functionality vs. Command Override

Some of SageTV's commands have default functionality that is built into the core program. When processing these commands via listeners, you can continue execution with the next-found listener (which could be in the core, or in the STV) by calling the "PassiveListen()" API function. To see a sample of this, look at the "Full Screen" listner's child widget chain: it sets the "LastSelection" variable to reflect the listener activated and updates the info line display area. It then calls "PassiveListen()" to continue the Full Screen command's default functionality. Enter the Full Screen command (Ctrl+Shift+F) to see both the info line being updated and the SageTV window toggling its full screen display status.

If you wish to override the built-in functionality for a command, simply do not call "PassiveListen()" in the listener's child widget chain. For this example, look at the widget chain below the "Home" listener. Usually, when you use the Home command, SageTV loads the menu defined by the "Main Menu" widget, however, in this example, Home has been overridden to simply update the info line and to pop-up a simple options menu telling you that the Home command was entered.

Conclusion

This ends the **Listeners** tutorial, where you learned the basics of using listener widgets to respond to user input. When done experimenting with any widgets in this menu, collapse the "Tutorial 13 – Listeners" menu widget tree.

Tutorial Set 14 – Hooks

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 14 – Hooks". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, information about the last hook action, and a couple buttons. (Close the dialog that pops up; it will be explained later.)

Before continuing, you may wish to review the Hook Widget and 5) Hooks – The Complete List references for a list of all hooks, the local variables they pass, and any return values they may expect. Unlike listeners, which are activated in response to receiving user input, such as a command, hooks are activated in response to some event that occurs in the SageTV core application. Like listeners, the child Process widget chain is executed when a hook activates.

Basic Hooks

The "RecordingScheduleChanged" hook shows a couple points about basic hook usage: it responds to the SageTV upcoming recording schedule being updated by updating the "LastHook" variable and the info line where it is displayed, and it pops up an options menu letting you know that the schedule was updated. If you let the SageTV window sit long enough, this hook will be activated. Or, if you are running another instance of SageTV or SageTVClient connected to the same server, you can set another show to be recorded to see this hook being activated.

For a few examples of hooks used below UI elements, check the widget chain below the "Button Focus Changes" panel. The buttons contained there have a couple hooks that update the info line when focus changes from one button to another, along with a third hook that marks the time when "Button 1" was last rendered. Use the Left and Right arrows to switch focus in order to see how the buttons and info line are updated.

As with the Process widget chains below the Listener widgets in a previous tutorial set, hooks used in a real-life STV could contain much more functionality. Check the default SageTV STV for examples of hooks doing something useful in response to the event that activated them. By the way: in the default SageTV STV, this hook would be used to update the SageTV window in order to reflect any UI changes that might be caused by the new schedule.

Hook-created Local Variables

Some hooks pass local variables that may be used in the hook's child widget chain to access whatever information may be needed to respond to the hook properly. When viewing the hook widget's properties, any such variable(s) are listed in parenthesis after the hook's name.

In the current menu, look at the properties for the "AfterMenuLoad" hook, where you will see the hook listed as: AfterMenuLoad(Reloaded). Thus, "Reloaded" is a variable passed to the hook's child widget chain. If you expand the drop-down box to see the list of all available hooks, you will see quite a few that pass such local variables. The sample menu uses this hook to pop-up a dialog to say whether the menu has been reloaded or not. In Studio, press F6 to refresh the menu; you will see that the menu has not been reloaded. Now, press Home to switch to the Main Menu, then use the Back command (Alt+Left) to return to this menu; you will see that the menu has been reloaded.

Hook Return Values

A few hooks require that the handling of that event result in the "ReturnValue" being set to indicate how to handle the event being reported, and possibly that some additional actions be taken – see 5) Hooks – The Complete List for full details. While the tutorial STV does not contain samples for any such hooks, you can see examples in the default SageTV STV. If you wish to see those examples, simply load the default STV and search for "ReturnValue". Or, expand and view the following hooks:

- DenyChannelChangeToRecord
- RecordRequestLiveConflict
- RecordRequestScheduleConflict
- RequestToExceedParentalRestrictions
- WatchRequestConflict

Conclusion

This ends the **Hooks** tutorial, where you learned the basics of using hook widgets to respond to SageTV core events. When done experimenting with any widgets in this menu, collapse the "Tutorial 14 – Hooks" menu widget tree.

Tutorial Set 15 – Themes

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 15 – Themes". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, several lines of text, and a few buttons.

Before continuing, you may wish to review the Theme Widget reference. Themes can be used to automatically apply properties and child widgets to other widgets to simplify defining the appearance of each UI element displayed by SageTV and to more easily keep that appearance consistent for similar elements. Themes are not needed for most display elements, but they help when various elements are to be displayed in a consistent manner. So far throughout these tutorials, themes have not been used except for the "Basic Menu Theme", which is used to set up some basic properties such as the background color for the menus and selected items. However, themes could have been used to reduce the amount of support code needed to display various items, such as buttons, as will be seen below.

Note: As with the other tutorials, this one will be introducing the basic concepts of the uses for theme widgets, but will not be providing an in-depth example of how to lay out all the themes for an entire STV. For examples of more complex themes, refer to fully developed STV files, such as the default STV used for SageTV.

Basic Theme Usage

The most basic theme usage can bee seen by looking at the one that has been used on every tutorial menu, the "Basic Menu Theme". A reference to this theme widget has been placed as a child of every menu widget. If you look at its properties, you will see the background color used for the menus and the color to be used for currently selected items. All other properties were left at their default values. Changing any of these properties will affect every tutorial menu. (If you experiment with changing these properties, don't forget to Apply the changes and to refresh the current menu's display by pressing F6.)

Another example of basic theme usage can be seen by looking at the text widget titled "This text is affected by its child theme" and its child widgets. Notice that the text color is different from the menu's description on the first line of text. The "Text's theme" widget defines that color. Also note that the text's layout is affected by the text widget child of the theme widget. Compare the properties of both of those text widgets to see that they differ. Check the "Ignore Theme Properties" option of the parent text widget, select Apply, then refresh the menu – you will see that the text is displayed in a different location and without a shadow.

A few points to learn from these examples:

- ☼ The theme widget applies its settings to its direct parent.
- ☼ The properties of the direct children of a theme widget are applied to those widget types that the theme applies to, unless the widgets have the "Ignore Theme Properties" option checked.
- ☼ If the theme properties are ignored for a widget, then all properties are skipped, even those that are blank. Note that the parent text widget left several properties as undefined, but those missing values that were defined by the text theme were not applied.
- ☼ Just like other widgets, themes can be reference widgets, so you can define a theme and all of its children in one place, then reference that entire theme in another location in the STV.

Theme Recursion

For most widgets, the properties defined on a widget that is a direct child of a theme widget will be applied to all the widgets affected by that theme. This is not the case for panel widgets, which have themed properties applied to them only if the theme is a direct child of that panel. The purpose of this is that you will very often want to use a panel to hold UI elements where additional panels can be used to break the display into sections. You will probably not want each child panel to use the same sets of properties, but it is desirable that the other UI display elements continue to use their themes to keep text or buttons displayed consistently.

Examine the widget chain below the "Panel Theme Example" panel to see examples of this. The easiest way to see which on-screen elements are affected by themed widgets or direct widgets is to highlight a widget in Studio and press F6 to refresh the menu.

The parent panel has a theme child with themed panel and text widgets. The parent panel also has other children for displaying the text in this example. Note that when you highlight the themed text widget, both lines of text are outlined in yellow in the SageTV window; however, when the themed panel widget is highlighted, only the section's parent panel is outlined. This shows that the text theme applies to text widgets being displayed, but only the parent panel is affected by the themed panel. Also, by highlighting the parent panel and then the "Child Panel" widgets, you can easily see that the "Child Panel" widget is not using the themed panel's Fixed Width property of 1.0.

You may wish to experiment with the property settings for the widgets in this example. While doing so, note that the only UI element widgets that have properties set are the ones that are children of the theme widget.

One last item of interest here is that this section's background color is set in the theme widget, while the border is drawn as a child of the parent widget.

Adding UI Elements via Themed Display Widgets

In the previous examples, the theme widgets had UI elements as direct children. Those themed child widgets were not displayed on the screen; only their properties were applied to other widgets that were displayed on the screen. What would happen if those themed UI elements also had child UI elements? In that case, those child widgets would be displayed in the SageTV window, as if those widgets had been added as children of the widgets affected by the themed UI elements. To see an example of this, see the widget chain below the "Added Child Theme Example" panel.

As you explore this widget chain, notice that the "themed panel" widget now has some children of its own. The section's border is now drawn as a child of the themed panel and the section's first line of text is defined in the theme. The themed text is still applied to all lines of text in this example, just like previously. In fact, the themed text also has a child shape widget which is applied to all affected text widgets.

The important points to remember are:

⚘ Widgets that are direct children of a theme widget become themed widgets whose properties apply to the widgets of that type affected by the theme. They are essentially templates whose properties are used for other widgets.

⚘ Widgets that are children of themed widgets are displayed as if they were contained in the UI widget chain below a Menu or OptionsMenu widget. They are not templates; they are actual rendered UI elements added as children of the widgets affected by the themed widget.

No Themes vs. Themes

To see an example of the difference between drawing items as buttons without using a themed item widget vs. using a theme to draw item widgets as buttons, see the last two panels in this menu: "Basic Buttons" and "Themed Buttons".

In the 'basic' panel, the only theme is used to set the text's normal and selected colors and shadows. Both buttons under this panel contain child widget chains which draw the SageTV v4 style buttons. If you remove the child widget chains from these buttons, they will not be displayed 'correctly'.

In the 'themed' panel, a themed button (item) widget contains a child widget chain to define a button drawing style. Note that the themed button has a child "Button text theme" widget chain to define how the button's text is drawn. (Remember: if the text widget had been added as a direct child of the item widget, that exact text would have been added to each item widget. Try it.) Once the button style theme has been defined, the buttons in this panel can simply be item widgets without any child UI widget chain. This would greatly simplify the goal of drawing buttons with a consistent appearance,

especially when you consider that this button theme, or elements of the theme, could be used throughout an STV, as is the case with the default SageTV STV.

More Examples: As mentioned above, see the default SageTV STV for more complex examples of themes used to define how various UI elements are drawn. Look for the "THEME ORGANIZER" menu, where several different theme styles are defined for various UI widget types.

Conclusion

This ends the **Themes** tutorial, where you learned the basics of using theme widgets and their child widget chains to define various settings to be applied to UI elements. When done experimenting with any widgets in this menu, collapse the "Tutorial 15 – Themes" menu widget tree.

Tutorial Set 16 – Animations

For this tutorial, make sure **Studio_Tutorials.stv** is loaded in Studio and then fully expand the tree for the menu widget titled "Tutorial 16 – Animations". While that menu widget is highlighted, press F5 to load the menu and view it in the SageTV window. You should see a blue background that fills the window, a brief description at the top, and a few colored boxes containing a line of text each.

Before continuing, you may wish to review the **Animation** property in the Properties Common to Many Widgets reference, and perhaps the Animation property vs. Refresh() API call tip.

Timed Animation

In this menu, examine the "Timed Animation" panel's widget tree. That panel defines a time period for the Animation property of "0,1000,0", which causes this entire section of the screen to be updated every second. When that panel updates, the displayed text is positioned according to the random position defined in the "Random location" panel. (This is the basic technique used for the SageTV screen saver – see the "Screen Saver" menu in the default STV.)

Dynamic Animation

Next, examine the "Dynamic Animation" panel's widget tree. That panel uses an Animation property of "=UpdateNow" to force an update of this section of the screen when that variable is 'true'. The text on this panel is randomly placed when the menu loads or when you select the text by clicking on it or by pressing Enter while that text has focus – the "X" & "Y" variables will be assigned random values and the "UpdateNow" variable will be set to 'true', causing the "Dynamic Animation" to update itself. During the update process, the "UpdateNow" variable is reset to 'false' so that the panel doesn't continue updating repeatedly.

Continuous Animation

Finally, examine the "Continuous Animation" panel's widget tree. That panel uses an Animation property of "=AnimationOn" to force an update of this section of the screen as long as that variable is 'true'. Every time the "Continuous Animation" panel updates, it updates its child "Incremental Location" panel, where you will see a RenderingStarted() hook. That hook will get called whenever its parent UI element is about to be redrawn. When the hook is called, the STV calculates the new X & Y values for where the "Incremental Location" panel is to be placed.

This animation will continue until you select the moving text, causing the "AnimationOn" variable to be toggled off. Select it again to start it moving again.

Conclusion

This ends the **Animations** tutorial, where you learned the basics of using the Animation widget property to update the display either on a timed basis, dynamically, or continuously. When done experimenting with any widgets in this menu, collapse the "Tutorial 16 – Animations" menu widget tree.

Example Set 1 – Customizing Menus

Note: For these examples of using Studio to change menu items in the default SageTV3.STV, make a copy of that STV and edit the copy. Remember to reset the read-only flag on the copy, if needed.

One of the most often stated desires of SageTV users is that they would like to rename, delete, reorder, move, or add menu items. Using Studio, this is a very easy task to accomplish. **Note:** The menus in SageTV3.STV are designed to hold up to 8 buttons. If you add more, the additional buttons will not be visible below the menu's display area inside the SageTV window, even though you can use the arrow keys to reach them. To simplify your menu edits, it is suggested that you keep the menus limited to 8 items or less.

Renaming a Menu Item

Perhaps there is some menu item that you think should be called something else. Maybe the "Program Guide" makes more sense to you as "TV Listings" or as some other term. This change can be done by:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted. The entire menu will be outlined in yellow, because the Main Menu widget is highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level.

3. Expand the "MainMenuContainer" panel one level.

4. Highlight the "Program Guide" item widget. Since it is the highlighted UI element widget, you will see it outlined in yellow in the SageTV window.

5. Press F2 to edit the widget's name. Type your preferred name for that button (perhaps "TV Listings", as mentioned above), then press Enter to accept the change.

6. Press F6 to reload the menu. Focus will shift to SageTV and you will notice that the button's name has been changed.

Removing a Menu Item

Removing a menu item is almost as easy as renaming one. For this example, the "Watch Live TV" button will be removed from the Main Menu:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level.

3. Expand the "MainMenuContainer" panel one level.

4. Right click on the "Watch Live TV" item widget and select "Expand Children" to fully expand the widget's child widget chain. (In this case, the widget only has a single child, but other buttons may have more children.)

5. As discussed in Tutorial Set 1's section covering Deleting Widgets, you should not simply delete the top-most widget that you wish to remove. In this case, that means you would not remove the "Watch Live TV" item widget first, because if you did, its child widget would become an orphan in the top-most level of the entire tree. Instead, highlight the child Action widget (with the "SageCommand()" API call) and delete it using the Del key on the keyboard.

6. Now that the only child widget has been deleted, you may delete the "Watch Live TV" item widget.

7. Press F6 to reload the menu. Focus will shift to SageTV and you will notice that the deleted button is no longer visible.

8. **Extra:** If you wish, you may use the Undo command (Ctrl+Z) twice to restore the deleted button, then reload the menu to see the restored button once again.

Reordering a Menu

Sometimes you may wish to simply change the order of the items on a menu. This example moves the "Program Guide" button (or whatever you may have renamed it to be) to be the top button on the Main Menu:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level. Then expand the "MainMenuContainer" panel one level.

3. Highlight the "Program Guide" item widget and move the item widget up by pressing Ctrl+U until it is the top-most child of the "MainMenuContainer" panel.

4. Press F6 to reload the menu. Focus will shift to SageTV and you will notice that the order of the buttons has been changed.

Moving a Menu Item to Another Menu

When rearranging the menus to suit you preferences, you may wish to move a menu item so that it is on a different menu instead of its original location, or you may wish to copy it so that it is in both menus. Let's copy the Weather menu item from the Online Service menu to the Main Menu:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level. Then expand the "MainMenuContainer" panel one level.

3. Scroll the Studio window down until you see the "Online Services Menu" menu widget. Expand that menu one level, then expand its "MainMenuContainer" panel one level.

4. Highlight the "Weather" item widget and Copy it by using the Copy command in the Edit menu that is on the menu bar at the top of the window. (If you then highlight another widget, the "Weather" widget will have a dotted outline around it.)

5. Return to the Main Menu in the Studio window and highlight its "MainMenuContainer" panel.

6. From the Edit menu, choose "Paste Reference", which will add an *italicized* "Weather" item widget as the last item under the "MainMenuContainer" panel.

7. Highlight *italicized* "Weather" item widget and Move it up to your desired location by pressing Ctrl+U as many times as needed.

8. Press F6 to reload the menu. Focus will shift to SageTV and you will notice that the Weather button is now visible on the Main Menu.

If you had wished to move the menu item instead of copying it (so that it was no longer in its old location), you would have used the "Cut" command in step 4, instead of the "Copy" command. Then, you would have used the "Paste" command in step 6.

Adding a Menu Item

Another menu customization might be to add a new menu item, one that isn't already available to copy or move from some other menu. The Schedule Recordings menu has a

button to "View Recording Schedule", but it only jumps to one of the styles for viewing the schedule (whichever one you configured it to be). Let's add menu items that jump directly to each of the available schedule display styles:

1. Go to the Schedule Recordings menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Schedule Recordings" widget highlighted.

2. Click on the symbol next to the Schedule Recordings's widget icon to expand the tree one level. Then expand its "MainMenuContainer" panel one level.

3. Add 2 new item widgets to the "MainMenuContainer" panel, then move them so that they are above the "Back to Main Menu" button. Name the two new widgets "View Interleaved Schedule" and "View Parallel Schedule".

4. Expand the "View Recording Schedule" item widget, above the ones that were added. Double click on its *italicized* child widget to jump to the "RecordingSchedule - SHORTCUTS" menu widget. Expand that menu completely, by right clicking on it and selecting "Expand Children".

5. From the newly expanded menu, left click & drag the *italicized* "*InterleavedSchedules*" reference to the new "View Interleaved Schedule" widget that you added in step 3.

6. Also from the newly expanded menu, left click & drag the *italicized* "*ParallelSchedules*" reference to the new "View Parallel Schedule" widget that you added in step 3.

7. Highlight the "Schedule Recordings" widget (same one as in step 1), then press F6 to reload the menu. Focus will shift to SageTV and you will notice that the new buttons are now visible. Select each one to see that they jump to the separate recording schedule display styles.

Adding an Entirely New Menu

Sometimes, changing an existing menu just isn't enough – you want to create a new menu to hold all of your favorite items. Let's add a new "My Custom Menu" to the Main Menu:

1. Start with a new copy of SageTV3.STV so that the changes from the above examples are not 'in the way'.

2. From The Widget Bar on the left, drag a new Menu widget to the middle of the Studio window. (Do not add it as a child of another widget.) There will be a new "Untitled" menu widget at the top of Studio's top-level tree – scroll all the way up to see it. Highlight that menu widget and rename it to "My Custom Menu".

3. Find the "Main Menu" menu widget in Studio and expand its tree one level, then expand its "MainMenuContainer" panel one level. Using what you learned in the previous examples, add a new button that links to the newly added "My Custom Menu" widget. (Add a new item widget to the "MainMenuContainer" panel, rename it to "My Custom Menu", move it to the desired position, then add a reference to the "My Custom Menu" menu widget as the item widget's only child. **Note:** At some point, the top level tree will be resorted, so the new "My Custom Menu" will no longer be at the top.)

4. In the Main Menu, highlight the "MainMenuTheme" theme and use Ctrl+C to copy it.

5. Return to the new "My Custom Menu" menu widget, highlight it, then use the Paste Reference command (Ctrl+Shift+V) to add a reference to the "MainMenuTheme" theme.

6. Add a text widget to the new menu and rename it to "My Custom Menu".

7. In the Main Menu, expand the "Main Menu" text widget one level. Use the Copy command on its child theme, "MenuTitle". Use the Past Reference command to add a reference to that theme to the text widget added in step 6.

8. In the Main Menu, highlight and Copy the "MainMenuContainer" panel. Paste it as a child of the new "My Custom Menu" menu widget, using Ctrl+V. (This creates a copy of the widget, not an *italicized* reference.)

9. In the Main Menu, expand the "MainMenuContainer" panel and Copy its child theme, the "MainMenuContainerTheme" widget. Use Paste Reference to add it as an *italicized* child of the "MainMenuContainer" panel in the new menu.

10. Copy the "Left" listener widget and Paste it to the new menu widget. Copy the Main Menu widget and use Paste Reference to add it as a *reference* child of the Left listener in the new menu.

11. You now have a blank new menu where you can add menu items, as described in the examples above. You may wish to start by adding a "Back to the Main Menu" button.

12. After you have added your custom menu items, don't forget to use F6 to reload the menu to see the buttons that were added.

Conclusion

This ends the **Menu Changes** examples, where you experienced the basics of using Studio to customize the menus to match your preferences.

Example Set 2 – Creating Pop-up Dialogs

Note: For these examples of using Studio to add a pop-up dialog using an OptionsMenu widget in the default SageTV3.STV, make a copy of that STV and edit the copy. Remember to reset the read-only flag on the copy, if needed.

There are times when you may wish to display a message or get input from the user while remaining at the current menu. To do this, the OptionsMenu Widget is used.

Adding an Informational Dialog

After creating your customized STV, you may wish to provide a way to let users know exactly what STV they are using – who created it and when. Let's add such a dialog to the Main Menu when the Info command is used:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level.

3. Drag a Listener widget () from The Widget Bar to the "Main Menu" widget. It will show up as a child of the Main Menu, named "Untitled". Right click on this new listener and select Properties. In the properties dialog, enter "Info: Display STV Version" (without the quotes) in the Listener field, then choose "Info" from the Listener Type drop-down list. Click on the OK button.

4. Drag a new OptionsMenu widget () from The Widget Bar to the newly-added listener widget. Highlight the new OptionsMenu widget, press F2 to edit its name, then enter "STV Version" and press Enter.

5. Scroll down in the Studio window to find the "THEME ORGANIZER" menu widget. Expand it one level, then expand the "OptionsMenuThemes" panel one level. Highlight the "OptionsConfirmTheme" theme widget and use the Copy command. (Ctrl+C, or Edit menu -> Copy)

6. Return to the newly added "STV Version" OptionsMenu widget, highlight it and use the Paste Reference command (Ctrl+Shift+V) to add the *italicized* "OptionsConfirmTheme" theme as a child widget.

7. Add a Text widget () as a child of the "STV Version" OptionsMenu widget and rename that Text Widget to something like: "Bob B. Robert's Custom STV"... or whatever you wish to call it. (Don't include the quotes.)

8. Add another Text Widget to the OptionsMenu widget, naming it with the version number and date of your STV, such as "Version 1.0, November 2, 2005". (Again, leave out the quotes.)

9. Add an Action widget to the OptionsMenu widget, naming it: ["Based on the SageTV STV, version " + STVversionNumber + " " + STVversionText]. (Include everything inside those brackets, including the quotes, but not the brackets themselves.)

10. Add a Text widget as a child of the Action widget that was added in the previous step. It does not need to be renamed.

11. Add an Item widget ( ) to the OptionsMenu widget, naming it "OK", without the quotes.

12. Add an Action widget as a child of the "OK" Item widget, renaming it to: "CloseOptionsMenu()", without the quotes.

13. Scroll down to find the "System Information" menu widget. Expand it one level. Highlight the "STVversionNumber" Attribute widget, then use Ctrl+Left-click to also highlight the "STVversionText" Attribute widget. Use the Copy command after both widgets are highlighted.

14. Return to the "STV Version" OptionsMenu widget, highlight it and use the Paste Reference command (Ctrl+Shift+V) to add two *italicized* Attribute widgets as children.

15. Press F6 to reload the Main Menu. Focus will shift to SageTV. While on the Main Menu, use the Info command (default: Ctrl+I).

Adding an Interactive Dialog

Some people would like to be able to exit SageTV from the Main Menu, instead of putting it to sleep. This can easily be accomplished using the Exit() API call, but to prevent accidentally closing SageTV completely, we will add a confirmation dialog, asking the user whether to proceed with the exit process. **Remember:** exiting a server instance of SageTV will prevent it from continuing to record. The client UI can safely be exiting without affecting any recordings.

The option will be added to the Main Menu as a new button which initiates the confirmation dialog:

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level. Expand the "MainMenuContainer" panel one level.

3. Add an Item widget (⬤) to the "MainMenuContainer" panel, press F2 to edit is name, then enter "Exit SageTV" (without the quotes) and press Enter. Use Ctrl+U to move it up one position, so that it is directly below the "Standby" widget.

4. Drag a new OptionsMenu widget (💬) from The Widget Bar to the newly-added menu item widget. Highlight the new OptionsMenu widget, press F2 to edit its name, then enter "Confirm Exit" and press Enter.

5. Scroll down in the Studio window to find the "THEME ORGANIZER" menu widget. Expand it one level, then expand the "OptionsMenuThemes" panel one level. Highlight the "OptionsConfirmTheme" theme widget and use the Copy command. (Ctrl+C, or Edit menu -> Copy)

6. Return to the newly added "Confirm Exit" OptionsMenu widget, highlight it and use the Paste Reference command (Ctrl+Shift+V) to add the *italicized* "OptionsConfirmTheme" theme as a child widget.

7. Add a Text widget (**b**) as a child of the "Confirm Exit" OptionsMenu widget and rename that Text Widget to: "Exit SageTV?". (Don't include the quotes.)

8. Add two Item widgets (⬤) to the OptionsMenu widget, naming the first one "Yes", and the second one "No", without the quotes. ("Yes" should be directly above "No".)

9. Add an Action widget as a child of the "Yes" Item widget, renaming it to: "Exit()", without the quotes.

10. Add an Action widget as a child of the "No" Item widget, renaming it to: "CloseOptionsMenu ()", without the quotes.

11. Add an Attribute widget (🔺) as a child of the "No" Item widget. Edit its properties: for the Attribute filed, enter "DefaultFocus"; for the Value field, enter "true" – both without quotes. **Note:** This tells SageTV to give the "No" button focus when the dialog is opened. If this attribute were not used, then the first Item, "Yes", would receive default focus... causing SageTV to exit if the "Exit SageTV" button were highlighted and "Enter" was pressed twice by accident.

12. Press F6 to reload the Main Menu. Focus will shift to SageTV. While on the Main Menu, you will now be able to select "Exit SageTV" to completely close the program.

Conclusion

This ends the **Creating Pop-up Dialogs** examples, where you experienced the basics of using Studio to create pop-up dialogs.

Example Set 3 – Adding a Customizable Option

Note: For this examples of using Studio to add a new option to the default SageTV3.STV, make a copy of that STV and edit the copy. Remember to reset the read-only flag on the copy, if needed.

Several customizable properties settings have been added to Detailed Setup -> Customize in the default SageTV3.STV. However, not all available properties can be configured there, so this example will show how to add a new option. **Note:** Not all properties can be configured in this manner, since some are only checked at program start-up time.

Adding a New Option to Customize SageTV's Settings

One of SageTV's recently added properties is "ui/reset_menu_history_on_sleep", which defaults to "true". If set to "false", this property tells SageTV to return to the menu that was active at the time when SageTV was put to sleep. Let's add a new option to allow setting this property from within the SageTV UI:

1. From the Main Menu in SageTV, choose Setup, go to Detailed Setup, highlight Customize, then select the top button on the right hand side (next to "Fast Forward & Rewind times"). While that dialog is open, use the Customize command (default: Ctrl_Shift+F12) to open Studio at that OptionsMenu widget. **Note:** You can return to SageTV to Close the dialog (we simply wanted to go to the section of the STV where all the Customize options are), then return to Studio.

2. In Studio, scroll down to find the "Expaned Record comamnd options" panel. (Yes, a couple words are misspelled! But, it is just the name of a panel and doesn't affect anything.) Highlight that panel and use Ctrl+E to expand all of its children. While the panel is still highlighted, use Shift+Left-click on the *italicized* "WideLeftRowPanelTheme" theme. Eight widgets should now be highlighted. Finally, use the Copy command (Ctrl+C) to copy all 8 widgets to the clipboard.

3. Scroll up in Studio to the "CustomizeItems" panel, which is the parent widget of the "Expaned Record comamnd options" panel. While the "CustomizeItems" panel is highlighted, use the Paste command (Ctrl+V) to paste a copy of all 8 widgets that were previously Copied.

4. Scroll down in Studio to highlight the new copy of the "Expaned Record comamnd options" panel, directly above the "VPagination" panel. Use F2 to rename the panel to "Reset Menu History", press Enter, then use the Up command (Ctrl+U) five (5) times, to move the panel directly below the original "Expaned Record comamnd options" panel.

5. Fully expand the "Reset Menu History" panel.

6. Rename the first Text widget child to: "Reset SageTV menus after sleeping".

7. Rename the Item widget to: "Reset Menu History". (This isn't necessary, but it makes the STV code more understandable.)

8. For the first Action widget, use F2 to edit the name of the widget, changing "Enabled" to "Yes", and "Disabled" to "No". Press Enter when done.

9. There are 3 occurrences of "ui/record_options_expanded" on two Action widgets. Change all 3 occurrences to "ui/reset_menu_history_on_sleep" by pressing F2 on each line and editing the existing text. Make sure all 3 uses of that property name are within quotes, just like the original property was. As above, press enter when done.

10. Press F6 to reload the Detailed Setup menu. Focus will shift to SageTV.

11. Scroll to the last option of the Customize section and select the button next to "Reset SageTV menus after sleeping" to change its setting to "No".

12. While SageTV has focus, use Ctrl+Z to put SageTV to sleep, then wake it again. You should return to the Detailed Setup Menu. (The Main Menu may be visible for a short time before returning to the Detailed Setup menu.)

13. You may now set the option to your preferred setting.

Conclusion

This ends the **Adding a Customizable Option** example, where you experienced adding a new option to the Detailed Setup -> Customize tab that enables configuring a SageTV property by toggling the current setting and saving it.

Extra: Explore the other options in the Customize section to see the ways in which some other properties are configured by using a pop-up dialog to ask the user what the setting should be instead of simply toggling it.

Example Set 4 – Adding a Basic Menu Animation

Note: For this examples of using Studio to add an animation to the buttons on the Main Menu in the default SageTV3.STV, make a copy of that STV and edit the copy. Remember to reset the read-only flag on the copy, if needed.

Adding a Fade In/Out Animation to Menu Items

In the default STV, the focused menu item is either highlighted, or it isn't. But, by using the RenderingStarted() hook, it is possible to adjust display parameters just before an object is updated on the screen. We will use this hook to perform a fade in or out for the Main Menu's items as they com into or out of focus::

1. Go to the Main Menu in SageTV and open Studio via the Customize command (default: Ctrl_Shift+F12). Studio will then open, with the "Main Menu" widget highlighted.

2. Click on the symbol next to the Main Menu's widget icon to expand the tree one level. Then, expand the "MainMenuTheme" Theme widget one level. Finally, expand the theme's child "ItemTheme" Item one level. (Below that item is just a single widget: the "SelectedRightEdgeButton" Theme.)

3. Add an Attribute widget ( ) as a child of the "ItemTheme" Item widget. Edit its properties: for the Attribute filed, enter "Shade"; for the Value field, enter "0" (zero) – both without quotes. Click on "OK" to close the properties editor.

4. Add a Hook widget ( ) as a child of the "ItemTheme" Item widget. Edit its properties: from the drop-down list box, select "RenderingStarted()", then click on "OK" to close the properties editor.

5. Add an Conditional widget ( ) as a child of the hook you just added. Highlight the new widget, press F2, and change its name to "Focused", without the quotes.

6. Add two Branch widgets ( ) to the "Focused" conditional widget, naming the first one "true", and the second one "else", without the quotes.

7. Below the "true" Branch, add a Conditional named "Shade < 1", then add an Action child of that Conditional and name it "Shade = Min(1.0, Shade + 0.04)", all without quotes.

8. Below the "else" Branch, add a Conditional named "Shade > 0", then add an Action child of that Conditional and name it "Shade = Max(0, Shade - 0.04)", all without quotes.

9. Go up a few lines to find the "SelectedRightEdgeButton" Theme. Edit its properties to change the Background Alpha and Background Selected Alpha properties to "=Shade*255", without the quotes. Also change the Background Image property to "MenuBarLong.png" and clear the Background Selected Image property. Click on "OK" to close the properties editor.

10. Go up a line to find the "ItemTheme" Item. Edit its properties to change the Animation property to "=Shade > 0 && Shade < 1", without the quotes. Click on "OK" to close the properties editor.

11. Highlight the "SelectedRightEdgeButton" Theme. (Just so all the Main Menu items won't be outlined in yellow after switching back to the SageTV window.)

12. Press F6 to reload the Main Menu. Focus will shift to SageTV. While on the Main Menu, use the Up and Down arrows to change which item has focus. Notice that the items now fade in & out.

13. As a result of editing the theme for these items, all the menus that use this theme also use this fade in/out process. From the Main Menu, select Setup to see this in action on another menu.

Conclusion

This ends the **Adding a Basic Menu Animation** example, where you experienced using the RenderingStarted hook to add a fade in/out animation to the items on the Main Menu and other menus.

9) Miscellaneous Studio Tips

Highlighting the Current UI Element

When using Studio, if the currently highlighted widget is visible in the SageTV window, that widget's UI display area will be outlined in yellow in the SageTV window. If the widget is part of a theme, it will outline all elements that it affects. Note: Shape widgets are not outlined in the SageTV window when they are highlighted in Studio.

Finding the Currently-Used STV Menu

While running SageTV, use the Customize command (default: Ctrl+Shift+F12) to cause Studio to open to the Menu or OptionsMenu widget that is currently active. Note: If Studio is already open, it will simply jump to the Menu or OptionsMenu's widget.

Action Chain Color Coding

All action chains are one of three types: Process, UI, or None. Action, Conditional, and Branch widgets are color coded by a small indicator dot in the upper left corner of the widget's icon to show the type of that action:

- ☯ **Process** – green.
- ☯ **UI** – blue.
- ☯ **None** – yellow.

For more details, see Widget Chain Types.

Run Multiple Instances

When comparing STV files or trying to duplicate a feature in another STV, it can be quite useful to run multiple Studio instances on a single computer. To do so, either run multiple instances of SageTV Client (using the "-multi" command line parameter), or run SageTV and SageTV Client. For each instance of SageTV, use the Customize command to launch Studio. Note that while you cannot currently copy widgets from one Studio instance to another, you can easily copy the text between windows via the copy (Ctrl+C) and paste (Ctrl+V) commands.

Copy Widgets from One STV to Another

As mentioned above, widgets cannot be copied directly between multiple instances of Studio, but there are a couple ways to copy widget chains from one STV to another:

1. Within a single instance of Studio, open the STV from which you wish to copy a widget chain. Highlight those widgets and use Ctrl+C to copy those widgets to the clipboard. Open another STV and paste the widgets into it. You may need to update any widgets that were secondary references in the source STV, such as themes.

2. Create a new temporary menu in the source STV. In that new menu, create references to any widget chains that you wish to copy to another STV. Next, export that menu (File -> Export Selected Menus) to a new STV file and import the new STV file into the destination STV where you wish to copy those widgets. Move the widgets to their new locations, delete the imported temporary menu. As in #1, you may need to update any widgets that were secondary references in the source STV, such as themes.

XML Widget Files

In the near future, SageTV will use XML files instead of STV files in order to add additional flexibility to how the SageTV Studio operates. For the meantime; there is a way to convert STV files to XML files and vice versa. This allows for editing of widgets in the XML file format, which may be desired for a variety of reasons. To save an entire STV's widgets as an XML file, just select File->Save A Copy As XML… in the Studio. Then select the XML file to save to. To convert an XML file back to an STV, do the following:

1. Open the Studio and select File->New and enter a filename that does not exist

2. Select File->Open XML… in the Studio and select the XML file you wish to convert to an STV file

3. You can then us the Save or Save As… commands in the File menu of the Studio to save the widgets to a selected STV file.

Edit Multiple Widgets' Properties at Once

If there are multiple widgets that are to use the same value for some properties, simply select all of those same types of widgets and open the properties dialog. Any changes will then be set for all selected widgets. Note: if any widgets have different values for some

properties, those property entry fields will be grayed out. Any changes to those initially grayed out properties will then be set for all selected widgets.

The Clock Display

If you want to change the format for the clock/time display, SageTV uses its own special-case clock attribute, as you may have noticed in the MainHeader for every screen.

This clock display will update itself on the minute. Other methods of creating and displaying the time or date will not automatically update themselves.

Some details on its usage:

- **$Clock** – Display the time only
- **$ClockTime** – Display the time only (same as **$Clock**)
- **$ClockDate** – Display the date only
- **$ClockDateTime** – Display the date followed by the time
- **$ClockTimeDate** – Display the time followed by the date
- **$Clock<format string>** – Everything after **$Clock** is used as a date/time formatting string according to the SimpleDateFormat rules in Java; see Java's SimpleDateFormat documentation for formatting codes at http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html

    1) Example: **$ClockE, MMM d** would displays **Tue, Dec 21**, if that were the date.

Note: The preformatted versions of $Clock adjust the data and time display format to match the system's internationalization settings. Using the <format string> version forces the use of a specific date or time format.

Animation property vs. Refresh() API call

It was noticed that Refresh() was being used a lot in custom STVs, so this will explain the Animation property a little more because its a lot more efficient than Refresh() and can achieve most of the same results. There is also some other auto-refreshing that occurs that you might not be aware of. Refresh() is instead to be used quite conservatively. It is used pretty rarely in the OriginalV2.stv.

First thing to point out is that you should not use Refresh() in the parent Action chains for Shapes or UI Widgets. Only use them in Actions that get fired from Listeners, Items (Select or click), Images (click) or Hooks (with the exception of the RenderingStarted hook – do not call Refresh() in an Action under this hook).

When you call Refresh(), SageTV will do a full evaluation of the Menu:

- ⚘ It will rebuild any changes in UI component hierarchy structure first (such as a tables children, but it reuses where it can).
- ⚘ Then it will go through and re-evaluate all of the data contents for all of the UI elements (parent Actions of UI components).
- ⚘ Then it marks the entire UI as dirty and in need of redraw. Then it will go through and perform the layout on all the dirty components (which is the whole UI in this case).
- ⚘ Then it will create the list of rendering operations used to perform the graphical update.
- ⚘ When it creates this list of rendering operations, that's when a Shape's parent action chain will be called.
- ⚘ The rendering operations get passed off abstractly to the rendering system which deals with DirectX9, Java2D and the 350 OSD. (the rendering system is plugin based)

The Animation property for Panels, Items and OptionsMenus has a different effect than Refresh() and is much more efficient. See the Animations property details, in Properties Common to Many Widgets.

When an Animation update occurs on a UI component, first it checks to see if it passes the UI conditional test (i.e. if it has a parent Branch or Conditional then it must evaluate to true). Then it checks to see if its time to perform an actual Animation based on the rules above. If it is time to perform an animation, then:

- ⚘ first it will mark its UI area as dirty
- ⚘ re-evaluate its UI children's parent Actions to refresh their data.
- ⚘ the rendering engine proceeds as above with the layout in that UI area and redrawing of it also

There is also 'refreshing' activity that occurs when focus changes or paging occurs.

Focus changes will cause the same behavior as Refresh() does, but only in parts of the UI hierarchy that are marked internally as 'focus listeners'. A UI component is considered a 'focus listener' if there is a GetFocusContext() Action in its parent Action chain or Shape rendering or conditionality, or if the variable Focused is used in its parent Action chain or Shape rendering or conditionality.

Paging changes will cause the same behavior as an Animation does, but only in parts of the UI hierarchy that are marked internally as 'paging listeners'. A UI component is considered a 'paging listener' if any of the variables IsFirstPage, IsFirstVPage, IsFirstHPage, IsLastPage, IsLastHPage or IsLastVPage are used in its parent Action chain or Shape rendering or conditionality.

What Text in the STV is Evaluated?

All "expressions" are displayed in Studio using a fixed width font, so any text displayed like that will be run through the expression parser and will be evaluated. Similarly, some widget names are used as variable names, such as TableComponent widgets, and will be displayed in a fixed width font also.

Difference Between Watch and WatchLive for Live TV

When **Watch(Airing)** is used to watch live TV, SageTV will play the airing via its usual "record to a file, then play it" Live Buffered mode. Playback will have a slight delay as the airing is first recorded to a file, then decoded and played. However, the user will be able to use all the usual transport commands such as pause, play, FF, REW, etc. The file size will grow as the recording continue; recording will not wrap back to the beginning of the file.

WatchLive(CaptureDeviceInput,PauseBufferSize) differs from **Watch(Airing)** in that when **PauseBufferSize** is 0, it will try to play the video without first encoding it to a file if SageTV supports such functionality for the specified capture device. In this situation, the user will not be able to use the transport control commands, since there is no playback buffer. If live playback is not possible, then SageTV will first encode the video to a file before playback, with the buffer file size set to a default value. (See below for more buffered file playback details.)

If **PauseBufferSize** is not 0, then SageTV will encode the video to a file before playback, with the buffer file size set to the specified value. The video will be encoded to the buffer file, wrapping back to the file's beginning when it reaches the end. Transport commands may be used, but it will not be possible to rewind further than the playback time that can be stored in the buffered file.

Use true as Conditional & Expressions as Branches

Since the **Conditional** and **Branch** widgets both simply contain expressions that are evaluated and compared, there is no rule saying that you have to have a detailed expression on the **Conditional** and simple **true/false/else** statements on the **Branch** widgets. All the expressions on both type of widgets are evaluated and compared, then the children action chains are executed for all **Branch** widgets whose expressions evaluate to the same value as the **Conditional**. So, it is possible to use **true** on the **Conditional** and more-detailed expressions on the **Branch** widgets. This allows a single **Conditional** + multiple **Branches** to be used instead of a series of **Conditional** widgets checking to see which of several situations is the active one.

Consequences of Conditional Expression Evaluation

As previously mentioned, all expressions are evaluated when using any conditionals and branches – this applies both to the use of the **Conditional** and **Branch** widgets and to the **If(Condition, true-expression, false-expression)** API call. In other words, SageTV does not stop evaluating branch expressions once it finds a match. So, a call to **If()** that uses API calls in all three parameters may not function as expected: **If(IsPlaying(), Pause(), Play())** will not toggle between paused and playing because all three of those API calls will be evaluated, not just IsPlaying() plus either Pause() or Play(). After evaluating all expressions, either the result of the first Pause() or Play() calls will be returned; Pause() or Play() do not get called again depending on the result of IsPlaying().

Using java Code

Java code can be used as part of an expression, but the usage differs from standard java notation: use an underscore ('_') instead of a dot ('.') for the package separator and do not use the '.' operator to access object instance fields or methods. To create a new class item, use "new_" in front of the statement. And, object member functions can be called by passing the object pointer as the first parameter of the function call. Examples:

- ♽ MyFile = new_java_io_File("C:\\file.txt")

- ♽ Use: java_io_File_isFile(MyFile)
  Instead of: MyFile.IsFile()
  (Note that the java object is passed as the first parameter of the function.)

Note: When adding java customizations, place the JAR files in the **jars** subdirectory in the **SageTV** directory and restart SageTV. When SageTV starts, it will append any JAR files found in that location to the JVM's classpath.

Calling SageTV API methods from Java

The SageTV API calls can also be invoked from Java code directly. To do this, you can use the following static method in the sage.SageTV class:

```
public static Object api(String methodName, Object[]
methodArgs) throws InvocationTargetException;
```

The methodName parameter should be the name of the SageTV API call and the method Args should be an array of the arguments to pass to the API call. It is safe to use null for methodArgs if the call has no arguments. If an exception is thrown during execution of the SageTV API call then the exception will be wrapped in an InvocationTargetException and thrown.

To compile code that uses this, just put the Sage.jar file in the classpath of your Java compiler.

A very thorough example of using the SageTV API from Java can be seen in the Webserver plugin developed by Nielm. The code for that project is publicly available on Sourceforge.net in the 'sageplugins' project

Translation Files

STV translations

SageTV can use language translation files to enable the UI text to be translated to other languages. To create the base translation file containing the text to be translated, use the **Generate Translation Source…** option on the <u>**Tools**</u> menu. This will create a file named "<STV base filename>_i18n.properties" in the same directory where the STV file is stored. In that file, each line will contain a text name, followed by an equal sign, followed by the text that you may wish to translate. Example:

S_Add_Favorite=Add Favorite

The translated language file should have a two-letter language code added to the filename, such as: "<STV base filename>_i18n_de.properties", where the code "_de" was added to indicate that it is a translation file for German. (The 2-letter code is specified by ISO 639.)

To translate phrases, change the text that appears on the right hand side of the equal sign. So, in the above example line, the phrase "**Add Favorite**" is the part to be translated.

The original translation properties file may contain much more text than that which needs to be translated, so you may wish to edit the file to remove some lines which do not really need translation.

Alternately, sometimes there may be a line of text that does need translation, yet has not been added to the translation properties file. In this case, you can add the line yourself and go ahead and translate it. To add another translation line, find the text in the STV to be translated and create a new translation properties file line for it:

1. Create a property name for the string. This is done by starting with the original text, then:
    a. Create a prefix to the line: an 'S_' for a Text, TextInput, or Item widget; or a 'D_' for an Action widget or an Attribute value. (The 'S_' prefix indicates that the text is static, while the 'D_' prefix indicates the text is dynamic and therefore the result of an expression.)

b. Add the original text, while replacing all non-alphanumeric characters with an underscore ('_'), but do not place more than one '_' character in a row

2. Add an equal sign
3. Create the translated text and place it to the right of the equal sign.

So, as an example, if there was an Action widget containing the following text, it would be converted to a translation property file line as:

Original Action widget text: `"MPEG Mode " + MpegMode`

Translation line: `D__MPEG_Mode_MpegMode="MPEG Mode " + MpegMode`

Core Translations

Some text is part of the core program and is stored in the sage.jar instead of the STV. If such text needs to be translated, extract the **SageTVCoreTranslations.properties** file from Sage.jar (by opening the jar file in a zip file viewer). Place this file in your SageTV folder.

Then, as above, add the 2-letter language code to the end of the base file name, such as: **SageTVCoreTranslations_de.properties**. Note that '_de' (for German) is just an example here; Sage.jar should already contain a '_de' translation properties file.

Again, as above, translate any text to the right of the equal sign on each line to your desired language.