



Architecture and Theory of Operation

The Savoy Difference

© Copyright 1995-2008 by Savoy WebEngines, Inc.

All Rights Reserved.

The Savoy WebEngine System software and documentation contain information that is protected by copyright. No part of the software or the documentation may be reproduced, photocopied, translated, stored on a retrieval system, or transmitted, in whole or in part, without the express written consent of Savoy WebEngines, Inc.

Savoy WebEngines, Inc., reserves the right to make improvements or changes to any and all parts of its Savoy software, at any time, without any obligation to notify any person or entity of such changes. The software described in this manual is furnished under a license agreement. This software may be used or copied only in accordance with the terms of the agreement.

Savoy WebEngines, Inc., makes no representations or warranties with respect to the contents or use of this manual, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Savoy WebEngines, Inc., reserves the right to revise this publication and to make improvements or changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Companies, names, and data used in examples herein are fictitious unless otherwise noted.

Savoy WebEngines, Inc., has attempted to supply trademark information about all company names, products, and services mentioned in this manual. Other product names mentioned herein are for identification purposes only and are hereby acknowledged if they are trademarks and/or registered trademarks of their respective companies.

Savoy WebEngines is a trademark of Savoy WebEngines, Inc.
Microsoft is a registered trademark of Microsoft Corp. Windows is a trademark of Microsoft Corp.

*Architecture and Theory of Operations
Release 4.0*

**Document Number: 0796-21
First Edition: July 2008
Printed in the United States of America.**

This document contains Copyright information belonging exclusively to Savoy WebEngines, Inc. and may not be redistributed or reproduced without permission.

Ubiquitous Rule Processing

It is generally assumed that all devices endowed with computers will somehow be able to communicate with each other – the industry has a sufficient number of proposed ways to do this [CEBUS, home network LANS, JINI, etc.]. Beyond *how* they will communicate, however, is the question of *what* they will communicate. That is, what is the logic, or set of algorithms, that determine overall behavior of a complex system of devices?

In some cases, a device may require specific services of another device. For example, an HVAC (Heating, Ventilation, and Air Conditioning) unit requires the specific services of a thermostat, a requirement designed into these particular devices. But suppose we wish to extend the ‘design’ of a device to provide an unforeseen interaction, like having your thermostat interact with a security system (reducing the temperature while away)? For this purpose, we need a way to extend the logic of a device.

In object-oriented computer languages (e.g., C++, Java), extending the design of an object is done by deriving another object from it and then adding methods to implement the extensions. However, if we added the extension as object methods, both participating devices would have to be ‘designed’ for the interaction; e.g., the security system would have to ‘know’ about thermostats. We have a stricter requirement: any device should be able to dynamically connect to any other device without any knowledge of their partners. To achieve this, we must impose some strict conformity to the extended methods.

Given this, a simple and compelling way to add logic to a device is to add standard methods that implement ‘rules’. These methods include ‘Assert’, ‘Evaluate’, etc., and are discussed in detail later. Since all device extensions conform to these standard interfaces, any device could, in principle, fire rules in any other device. The determination of which rules to fire in which devices is not part of the device implementation, but rather encoded into a common rule data structure.

Many devices have ‘implicit’ rules – a thermostat has a rule that says ‘when temperature is below the set-point, send message to turn on the heater’. With embedded computers inside the thermostat, we can enhance its behavior to also send a message to the security system (or whatever) – we call this an ‘explicit’ rule.

Adding explicit rules to a device can be implemented in one of several ways depending on the proximity of the processor to the device. If the processor is embedded within the device, rules can be compiled and downloaded into it. If the processor is a controller that connects to one or more devices, rules can similarly be loaded into the controller. Finally, if neither an embedded processor nor a programmed controller is appropriate, the rules can operate in a general-purpose server computer.

The advantage of a common rule structure across all of these implementations is (1) they can all compatibly interact with each other, (2) it is independent of the communication technique and can arguably utilize any communication facility

available (particularly any standard that emerges), (3) does not require a common operating system nor implementation language to run in the processors, and (4) encodes the actual functionality of the interactions so that the design of devices themselves can remain generic.

We present here an overview of the fundamental concepts of how we model a device and how multiple devices interact through rules.

Device State

Devices are assumed to have a finite state. The total state of a device is the enumeration of all possible internal configurations or modes that the device can acquire. The entire system is therefore subject to the formal modeling techniques of finite-state systems.

Note: An important aspect of the WebEngine is the adherence to a finite-state model. Here, we concern ourselves with state transitions as opposed to (say) messages that cause events to happen.

Assertions

The transition from one state to another is the result of an 'assertion'. Assertions do not have direction: an assertion *to* a device that 'causes' a state transition is indistinguishable from an assertion *from* a device indicating the 'effect' of a transition.

Conditions

Closely related to an assertion is a 'condition'. Conditions test whether a device is in a particular state or not and have a corresponding value of True or False.

Rules

By combining conditions and assertions, we can construct simple rules that describe how two or more devices should interact. The rule has three components; a list of conditions, and two lists of assertions -- a 'True list' and a 'False list'.

Rules have a simple state, either True or False, which is determined by the AND of all conditions in the condition list. Activity occurs when rules change state. When a rule goes from False to True, the True list is asserted, and when the rule goes from True to False, the False list is asserted. This rule construct has been shown to be simple, convenient, and sufficient with respect to describing most device interactions.

The logical OR-ing of conditions is accomplished by multiple rules, since all rules operate in parallel. Practice has shown that this approach is easy to understand and encourages good design principles thereby minimizing design errors.

Rule processors implemented in embedded devices have the constraint that the conditions of a rule must all refer to local state. Assertions, on the other hand, can be global to all scoped devices.

Rule processors implemented in server computers have no such constraint. In these systems, interconnected servers create proxy devices for all scoped

remote devices. Consistency between these proxy devices and the remote real devices they represent is automatically maintained, permitting conditions to appear in rules (actually, it is the conditions of the proxy devices that are being tested).

Device Name-scoping

An important objective of this system is scalability and to achieve it we must eliminate the geometric growth in complexity associated with large numbers of devices. Name scoping permits the interaction of a practically unlimited number of devices without the encumbrance of geometric growth in complexity. It does so by permitting the construction of device-name hierarchies that requires that devices interact on an as-needed basis, in much the same way that computer languages scope variable names.

Devices interact according to whether they are in-scope or not. Scope [see section Device Name-scoping, page 4] determines the propagation of assertions across multiple interconnected processors. Devices that share a common scope are said to be 'in-scope' and fully interact according to prevailing rules.

Processing Element Categories

Rule processing is implemented in several different ways depending on the form of the processing element. To date, several distinct forms have emerged:

- Networked Computers having general I/O capabilities
- Embedded Attached Computers having compiler runtime capabilities
- Embedded Micro-processors having interpreted runtime system

From these definitions, there are three corresponding types of rule processors:

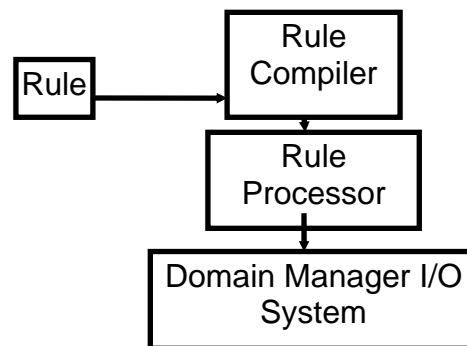


Figure 1 Rule Compiler, Rule Processor, with Domain Manager [see below] Architecture

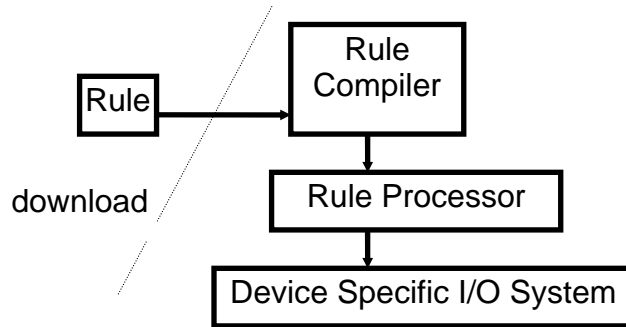


Figure 2 Downloaded Rule Compiler and Rule Processor Managing Specific Device Types

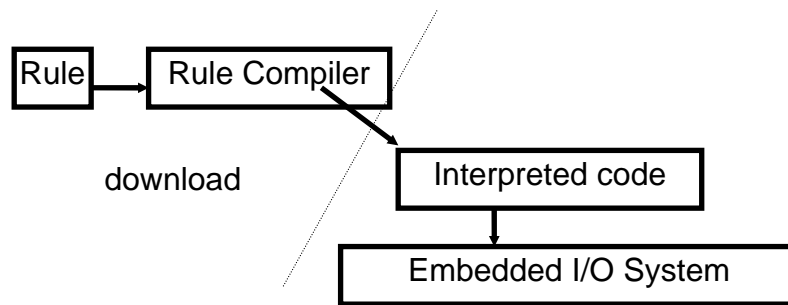
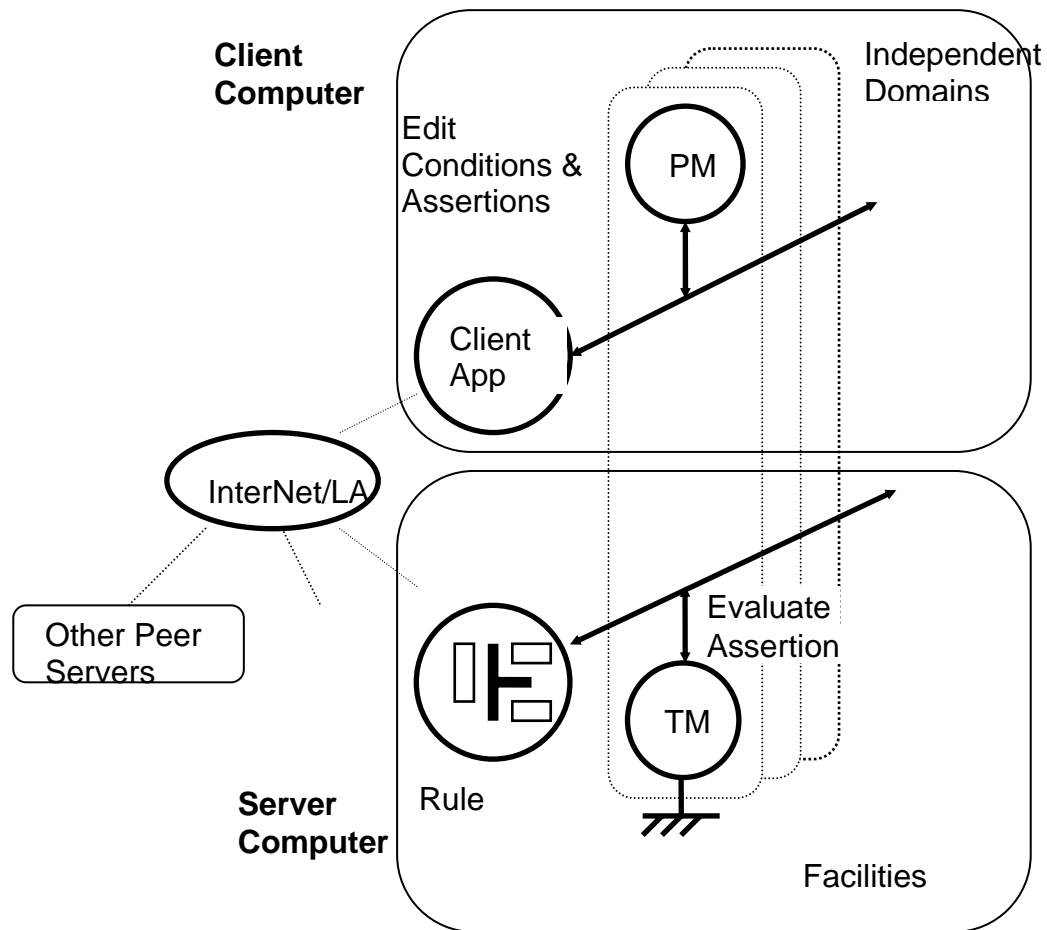


Figure 3 Server-based Compiler, Rule Processor, Specific Device

Component Organization

The Savoy WebEngine is organized into two principle types of components:

- computing elements that implement device behavior in the form of a server module, an attached embedded controller, or an embedded micro-processor within a device.
- a set of Client modules for development or viewing.



Multiple Servers can be linked together, functioning in a peer-peer mode. In general, the Server contains all of the logic to control devices, including the execution of a rule-based logic engine.

Client Applications

The Client applications include the capability for rendering and analyzing device behavior in both spatial and temporal views. Spatial views are physical layouts of an environment for the purposes of monitoring and control, while temporal views are a time-line history of causes and effects required to determine what happened and when. The Client applications also host application components that install

and configure devices and an editor for creating rules that automate their operation.

For the WebEngine, the advantage of this network organization is twofold: first, the set of connected Servers can be running continuously, whereas the Client applications need only run when the user needs to access the system, and second, the Client applications can access the Server(s) over networks. Actually, a Client application can connect to multiple Servers, and at the same time, a Server can be connected to multiple Clients. Savoy WebEngine Release 2.0 and beyond uses Internet TCP/IP protocols between the Server(s) and all Clients allowing access over the Internet.

Description of interoperation of CCTV viewing and storage, Access Control events and camera or multiplexer drive of pan, tilt, zoom operation.

Savoy WebEngines can be co-located to a centralized demarcation and connection panel for all camera feeds or WebEngines may be distributed around a building or campus of buildings lessening the requirement for camera cable runs by utilizing the network for transport of field analog/digital conversion at geographically dispersed WebEngine concentration centers.

Devices and systems such as camera (a device), sensors from intrusion detection systems (such as fence zones, swimmer buoys, security and fire systems) and access control systems providing event notification are all connected to WebEngine serial ports or through software API's so that full duplex communications occurs at the WebEngine software platform. The WebEngines therefore have visibility into the events and operations of these diverse systems and generate reactions and events to other systems, devices or Client applications.

In the case of an intrusion detected by an Access Control System component (i.e., Access Denied), a security device (zone state change to 'off-normal') coupled with other system events and their respective states can and will cause pan, tilt zoom camera(s) to drive to pre-set positions that are correlated with the event in progress. This real-time defensive intrusion process is managed by Savoy's rule processor which is the functional software element in WebEngines (servers) and Consoles (clients). The Rules Processor runs a real-time event driven process where one or more 'triggered' events will cause some number of operations to occur both locally and globally. Therefore, in addition to driving one or more cameras to pre-set fields of view, the WebEngine's rule processor can and will cause dynamic connections to Consoles with content delivery including

audible announcement, text display, log to file, site plan display articulating point of event, as well as providing a dynamic composition of video images across one or more video displays permitting the real-time risk assessment by both local and remote Security Operation Center (SOC) workstation personnel. SOC workstation personnel will have the system generate possible operator actions as well as permitting a free text log of the operator's process.

Savoy currently delivers Video WebEngines with three compression algorithms fully operational. They are wavelet, JPG, and H323. We have optimized the utilization of each of these compression schemes to provide best quality recorded video, dynamic bandwidth management, asynchronous live viewing, and full duplex audio/video broadcast capabilities. Savoy has generated applications that make use of these algorithms to provide the most efficient means of event driven live processing as well as providing efficient retrieval and auditing of recorded video with the best possible image content and a fast means of drilling down through hours or days of non-descript events both locally and over networks.

Savoy supports all Pelco PTZ cameras (direct drive), multiplexers and matrix components (Coaxitron drive) as well as Panasonic cameras and multiplexers. Delivery of any additional manufacturer system functionality is typically two to three weeks and is dependent on the willingness of the camera manufacturer to provide protocol and related SDK tools.

Savoy continues to engineer digital video solutions that provide enhanced viewing, recording, retrieval, and network transport capabilities. Integration of diverse field systems and devices coupled with next generation image processing technologies and Web based quality of service capabilities position Savoy with a substantial technological lead in the IT and Physical Security markets. Savoy's underlying technology and systems architecture permit the rapid assimilation of new technologies into the WebEngine software platform.

A door alarm (whether software generated or hardware driven) will interface with Savoy either through wired logic, serial or API communications. Regardless of the interface path, Savoy treats all device state changes as events. The Savoy Rule processor then determines if the state change is sufficient to trigger an action. If the event is true then Savoy generates a transaction number and associates this with the video files containing the date, time and camera. A variety of fields can be designed to handle specified system transactions including access control, point-of sale, security systems, and operator generated events to name a few.

Savoy employs a distributed database architecture permitting scalability to thousands of WebEngines in a typical system design. Absent a physical event trigger from a third-party field device Savoy generates video files locally and classifies them as either 'normal' or 'motion' events. 'Normal' events are typically held for a short time period (say 12 to 24 hours) where 'motion' events are locally

archived in a circular storage scheme for days, weeks or months. The storage interval is determined by the size of storage media, connected camera population, required frame storage rate, and the frequency and volume of motion activity from cameras and events generated by foreign device triggers. Any specified event (one that the Rules Processor has been set to 'watch' for) is tagged and its existence is immediately reported to a Console either locally, globally or both. The Console operator upon receipt of this real-time event can either manually or automatically preserve the 'off-loaded' video content along with any other data Savoy was instructed to push at one or more Centralized massive storage archives.

The Savoy Difference

Scalable Network Video

As of release 2.6, Savoy has introduced an advanced feature, termed Scalable Network Video, designed to optimize the use of network bandwidth for the delivery of real-time video from Savoy WebEngines to one of more Savoy client Console applications. This document describes the techniques employed to achieve these improvements.

Previous releases of the Savoy WebEngine included 'Adaptive Network Management'; a technique whereby the load placed on the network is automatically regulated as a function of the 'ambient' load imposed by other network applications. As the ambient load increases, Savoy's demand decreases accordingly. Scalable Network Video is a major improvement over Adaptive Network Management.

Scaled network video uses JPEG compression and is only applied to the delivery of real-time video to remote clients: video stored on the server's disks is always in the form of high quality wavelet compression. Further, when a remote client requests archived events (via either file transfer or frame-by-frame), the delivered video is high quality wavelet compression.

Scalable Network Video dynamically adjusts the compression encoding 'efficiency' for each video frame delivered to each connected Console in order to optimize network load. To accomplish this, each Console is in constantly providing Window and Matrix sizes to each connected server. At the server, this information is used to compute the compression level that produces the **minimum** size of each frame, such that when rendered on the requesting client, it appears to have acceptable quality. By computing the minimum compressed size dynamically, the system applies minimal load to the network.

With scalable video, the load on the network is directly dependent on the viewing requirements imposed by the user at the client. For example, a Console application displaying small video windows will use less network bandwidth than will a Console with large video windows. If the user expands the size of an active window, the network load will likely increase. The use of Video Matrix windows will change network demand depending on which configuration is selected. In all cases, the delivered video sizes are the minimum required for good quality viewing.

Note that each Console is treated independently; one Console viewing video in a large window will not affect the video delivered to another Console viewing it in a smaller window.

Savoy Console Application

While there are many products on the market that deal with remote digital video surveillance, most are designed in a conventional way described as a 'user driven tool'. In these, like most computer applications, the user performs a series of steps such as:

- Opens a connection, file, or configuration, etc.,
- Views/edits/analyses/queries the opened connection/file/configuration,
- Closes the connection/file/configuration

In these scenarios, the user is directing the activity and basically involved with a single entity, like a single set of remote cameras.

In contrast, the Savoy system is not a conventional computer application; rather, it is an 'active environment' that permits a high degree of parallel activity. Much of this activity can be directed by the user, as in conventional systems, but as important is the activity that is automatically managed by the system itself.

A good example of this is that remote WebEngine servers can **dynamically** connect to a Console based solely on activity at the remote site. The remote rule-based server can detect an alarm condition and then 'notify' one or more Consoles requesting a connection, voice annunciation, video matrix configuration, and more. In these cases, the Console operator observes that the system automatically accommodates the new connection by re-formatting the video layouts (tiling the existing windows to make room for the new ones), adding buttons, maps, and so on – all without operator intervention.

Further, the operator can make as many connections to remote servers as desired, viewing all of them concurrently. The operator can fetch remote video files, view live images, issue queries to remote databases, and more, all concurrently.

All the while, the Savoy Console is in direct communication with all remote servers, responding immediately to their activity. Even remote servers that aren't connected are actively 'supervised', making sure that the LAN/WAN link is operating in case it is needed.

How is this accomplished? The Savoy Console consists of a set of configurable 'plug-ins' that operate as individual applications, much like the computer's operating system. The Console plug-ins, however, are designed for close interaction with each other, and they are all designed to operate concurrently to each other. Many are designed to deal with multiple contexts, such as the video display plug-in.

Plug-ins can be included/excluded in the Console application through the use of a Setup application. By including only the essential set of Plug-ins, your Console becomes a very lightweight application that might run on your laptop computer. Conversely, by including an extensive set of Plug-ins, the Console can operate a complex, multi-display operations center – the same application spanning a wide range of uses.

New Plug-ins are constantly being developed and improved. Once released, they become available for inclusion in any Console application. Consequently, the Savoy Console is not only an 'active environment', but also one that will grow to accommodate future needs.

What follows is a detail description of how these advantages are exploited to produce product differentiation and competitive advantage.

Camera Viewing

Typical competitors can view a maximum of 16 cameras at a time, on a single computer monitor.

Savoy has the ability to view an unlimited number of cameras from an unlimited number of remote sites (see below) and display them on an array of up to 16 computer monitors. Further, the Savoy Video Matrix feature can render combinations of cameras from any site and display them in a 'multiplexed' form of 1x1, 2x2, 3x3, 4x4, 5x5, 6x6 camera arrays – features typically found in expensive hardware multiplexors. Savoy supports up to 10 of these Video Matrix views (they can be independently displayed on separate computer monitors if desired), and each Video Matrix can have up to 10 configurations (selections of cameras/array configurations). All of these are easily selected by typing a Function Key and a few numbers.

Video Matrix views can also serve as a 'spot' monitor whereby the user desires to immediately assign a particular camera to one of 5 monitors. The user's right-click on any camera window easily makes the assignment with no configuration required.



Figure 4: Savoy's Video Matrix feature

Multi-site Integration

One competitor integrates multiple sites by dragging/dropping resources (cameras) from a list of all network servers onto a single screen, limiting the total number of cameras to sixteen. After placing all of the desired cameras onto the screen, the user can name and save the configuration. A list of configurations is provided in a pull-down menu for rapid reloading later.

Suppose the customer has 10 sites that they want to monitor, and let's suppose that they want to be prepared to rapidly view any two sites at a time. So, they need to create a configuration for sites 1&2, and sites 1&3, and sites 1&4... and 2&3, 2&4... 3&4, 3&5... and so on. How many in all? Do the math and you'll find that for N sites, it requires $N(N-1)/2$ configurations to view any two at a time. For $N = 10$, the total number of configurations that need to be designed is 45. For 20 sites, the number of configurations is $20(19)/2 = 190!$ And, of course, it gets worse (as N square) as the number of site grow.

In contrast, Savoy's Console is similar to an operating system that is managing many independent applications. Savoy calls these applications 'Plug-ins', and the important point is that they all operate concurrently. This permits the Console to independently view as many sites and in any combination the user requires. Typically the Savoy Console displays as many as 64 cameras showing real-time video from a potentially unlimited number of sites.

Site Connection

Another competitor connects to remote sites by adding the IP address to a simple list and then including cameras from that site onto a configuration.

Savoy has several additional features: first, the ability to identify named sites as having a group of servers (Savoy WebEngines), permitting the user to connect to the group collectively and at once. Consequently, if the number of servers at a particular location changes, this change is transparent to the user. Secondly, remote sites can dynamically cause a connection to the Console as the result of some event, like an alarm condition. Third, this event can cause multiple remote Consoles to connect (say one at an office, another at a police department, or monitoring service).

Management Capabilities

Many competitors have the ability to detect failed cameras and report them to a local security panel using wired logic, which in turn, can send a signal to a monitoring service indicating that a camera from a group or bank of cameras has failed. They cannot indicate the specific camera but, rather only the group which could be a group of as many as 32 cameras.

Savoy, on the other hand, is in constant communication with all of the servers (WebEngines) through a web-hosted database. Users can at any time connect their Browser to their private view of the database (password protected) and view the current status of their system(s). A separate service monitors the status of all sites and sends notifications to the user by any of several techniques (email, paging...) without any intervention from a monitoring service. Of, course, if desired, Savoy's server can also contact a monitoring service directly. The resolution of content managed by Savoy's Web Based Service provides notification of failure specific to the individual camera. This content is delivered to portable web enabled devices such as PDA's, cell phones and more.

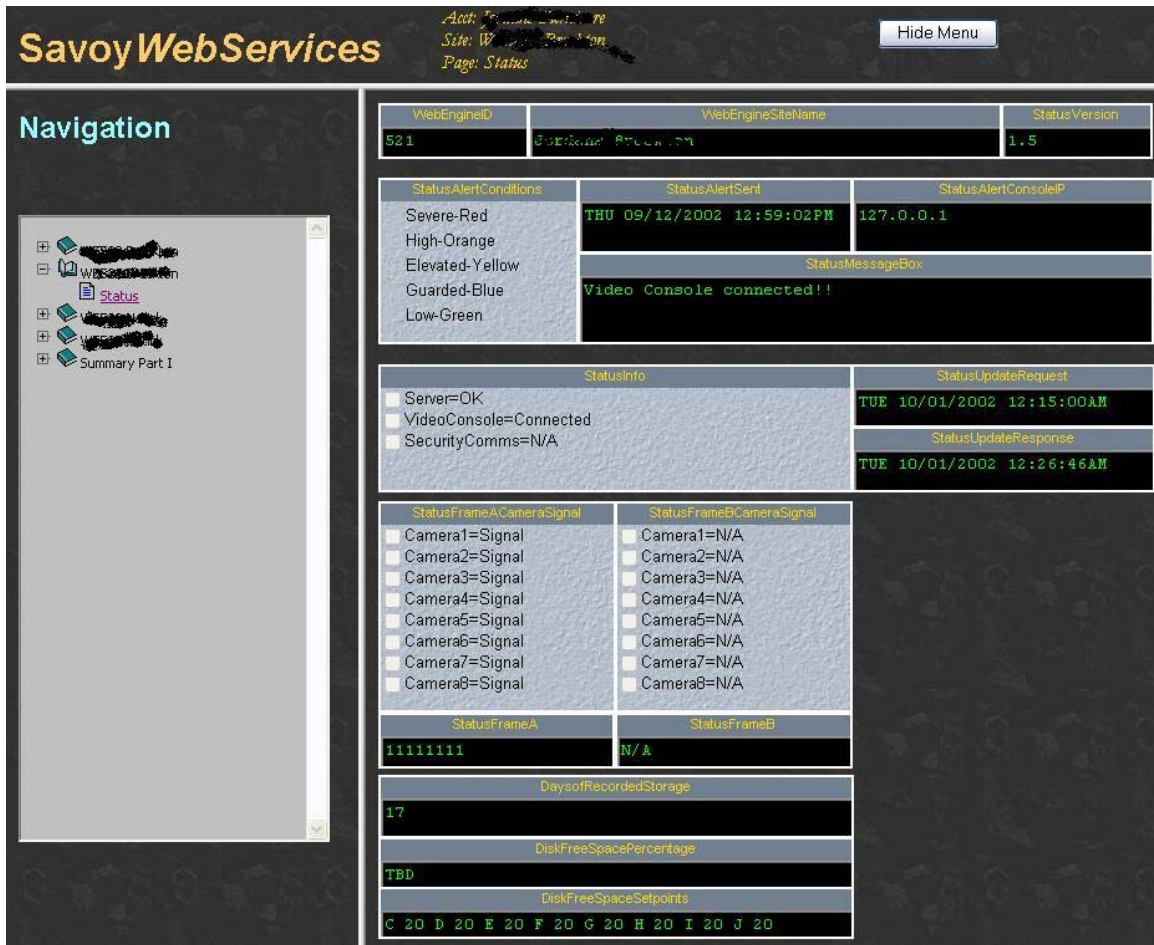


Figure 5 Browser View of Savoy's WebServices Customer Site

Further, the Console provides supervision over any number of sites. What is supervision? Suppose a customer has 50 sites that at any time could be connected to the Console. How does the user know that, if a connection were necessary, it would all work? Is the server operational? Is the Internet available? Supervision actively monitors all supervised sites for their preparedness to make a connection. If any are unable to do so, a status indicator on the Console notifies the user.

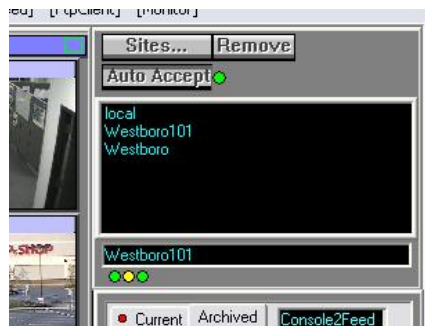


Figure 6 LEDs show the Status of Unlimited Number of Supervised Sites

Searching and Retrieval

Competitors can perform remote searches in a variety of ways and retrieve them. They have the capability of viewing them during retrieval and the files can be saved in several file formats including AVI.

Savoy can also retrieve video segments and save them as named events. However, for viewing the archive, Savoy offers QuickView.

QuickView delivers archived video immediately, without the delay of a file transfer, and renders them frame-by-frame on the Console. QuickView has a slider that acts as an accelerator, permitting the user to determine how fast the files are traversed. When the slider is set to the 'slowest' setting, QuickView delivers every frame for the selected camera from the remote server. Place the slider in the 'fastest' setting, and QuickView delivers only a single frame from the selected video segment (a segment is typically 60 seconds of recorded video). Obviously, when the setting is in anything but the 'slowest' setting, QuickView is delivering a subset of the stored video. Ideal for searching a vast period of time for many queries (such as 'when did that briefcase disappear from that table?'), QuickView can rapidly traverse stored video at a rate of only a few minutes per hour of search interval. Once the segment is found wherein the briefcase first disappears, the user simply moves the accelerator to increase the frame content until the targeted point in time is reached and full resolution of the video is playing and made ready for permanent archival.

Unique Features

Savoy WebEngines, offers numerous features that are truly unique. To name a few:

- Ability to dynamically assign any camera to a live H323 compression standard that delivers high frame rate and full duplex audio
- Ability to configure Touch Panel windows with various display components (buttons, bitmaps, etc.) for advanced user interaction with the servers.
- Ability to dynamically assign a Console to remote client computers (like at a monitoring service) upon any event or operator command
- Ability to have any number of Consoles concurrently connected to any number of servers.
- Ability to integrate with the diversity of capabilities supported by the Savoy WebEngine platform.