Module 17

Introduction to the Sausalito Architecture

Objectives

After completing this module, you should be able to:

- Explain the Sausalito software architecture design goals
- Create and extend object schemas
- Use the Sausalito `cceclient` utility to activate systems changes from the command line
- Analyze Cobalt Configuration Engine (CCE) server log files
- Explain the contents of the `/usr/sausalito` directory tree
- Extend the Server Desktop with menu items, options, and pages
- Create languages for the Server Desktop using the I18n component
- Create and implement a new Server Desktop style

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- The *Sun Cobalt™ Qube 3 Developer's Guide,* which is available at `http://developer.cobalt.com/sausalito/index.php`
- For more information on PHP the widely-used, general-purpose scripting language that is especially suited for Web development and can be embedded into HTML see: [`http://www.php.net/PHP`]
- For more information on gettext' utilities which are a set of tools that provides a framework to help produce multi-lingual messages [`http://www.gnu.org/software/gettext/gettext.html`]

What Is Important in This Module?

This module describes the Sausalito architecture and how to use it to extend the server appliance's out-of-the-box service features. The major topics in this module are:

- Sausalito Architecture Overview
- Sausalito Software Architecture
- SET Transaction Overview and Components
- SET Transaction Configuration
- Registering an Event Handler
- CSCP Communication
- GET Transactions
- Server Desktop Screen Generation
- Navigation Sub-system Overview
- Internationalization (i18n)
- User Interface Foundation Classes (UIFC)

As you read this module, look for answers to the following questions. See Appendix A for answers.

1. What is the Sausalito software architecture designed to provide?

Answer: ____________________

____

2. What are the Sausalito processes?

Answer: ____________________

____

3. What are the main components of CCE?

Answer: ____________________

____

4. What are SET transactions and how are they triggered?

Answer: ______________________________

____

5. What is the purpose of the Cobalt System Configuration Protocol?

Answer: ______________________________

____

6. How is an Event Handler registered?

Answer: ______________________________

____

7. What are GET transactions?

Answer: ______________________________

____

8. What is the purpose of the Navigation sub-system?

Answer: ______________________________

____

9. What is the function of Internationalization (i18n)?

Answer: ______________________________

____

Overview

Sausalito is the name of the Sun Cobalt Qube 3 Plus server appliance's software architecture, which is designed to provide an extensible development platform that enables third parties to customize and modularize the platform and quickly implement services, applications, and languages.

Having a clear understanding of Sausalito is helpful for maintaining the server appliance. This module describes the Sausalito architecture and the process for extending the server appliance's out-of-the-box service features.

Figure 17-1 shows that the Sausalito software layer lies between the Server Desktop, applications, and the Linux operating system.

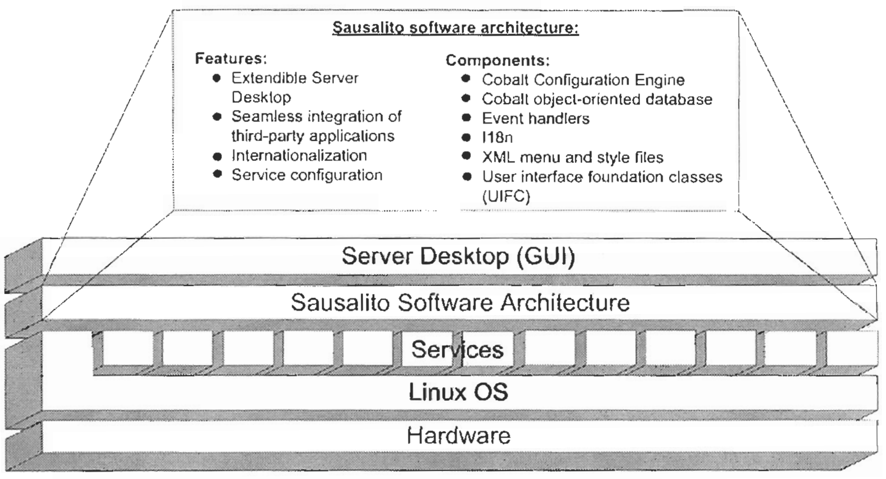

Sausalito software architecture:
Features:
Extendible Server Desktop
Seamless integration of third-party applications
Internationalization
Service configuration
Components:
Cobalt Configuration Engine
Cobalt object-oriented database
Event handlers
I18n
XML menu and style files
User interface foundation classes (UIFC)
Server Desktop (GUI)
Sausalito Software Architecture
Services
Linux OS
Hardware

Figure 17-1 Sausalito on the server appliance platform layers

Sausalito Handles System Changes

Sausalito handles operating system modifications, such as creating, deleting, and modifying users and groups. Sausalito also handles requests for changing system functionality, such as enabling SMTP, CGI scripts, and Microsoft FrontPage server extensions. Having Sausalito handle these requests helps hide the complexities of the operating system from the end user.

Sausalito's fully open Application Programming Interface (API) allows programmers to seamlessly integrate additional applications, giving them the look and feel of the server appliance.

Object-Oriented Programming

Sausalito is based on object-oriented programming; everything in the system is an object that can be added, removed, changed, and extended. Object-oriented programming helps Sausalito live up to its goal of extensibility.

Figure 17-2 shows a block diagram of the Sausalito software architecture. This diagram expands later in this module to show more details about Sausalito's components and process flow.

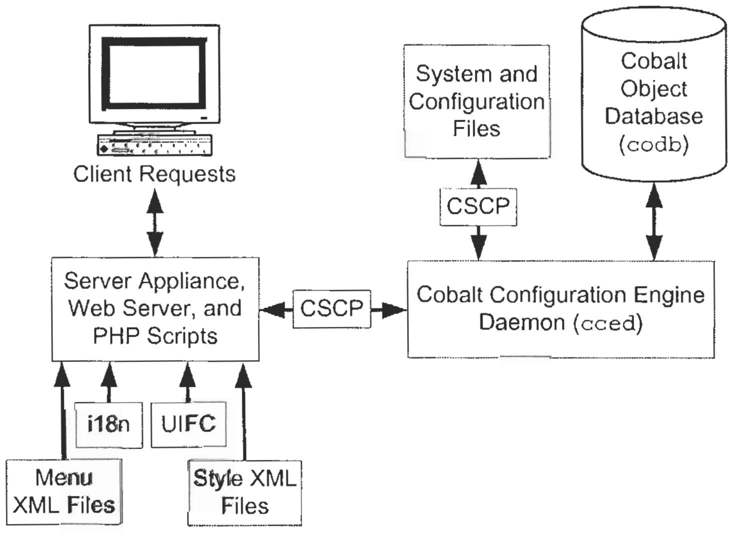

Client Requests
System and Configuration Files
Cobalt Object Database (codb)
CSCP
Server Appliance, Web Server, and PHP Scripts
CSCP
Cobalt Configuration Engine Daemon (cced)
i18n
UIFC
Menu XML Files
Style XML Files

Figure 17-2 Sausalito software architecture block diagram

Sausalito's Two Main Processes

Sausalito can be broken down into two main processes: SET transactions and GET requests. These two processes let you manipulate the system software through the Server Desktop, while keeping the system software and Server Desktop logically separate. Figure 17-3 shows the major Sausalito components and the process flow for the two main processes.

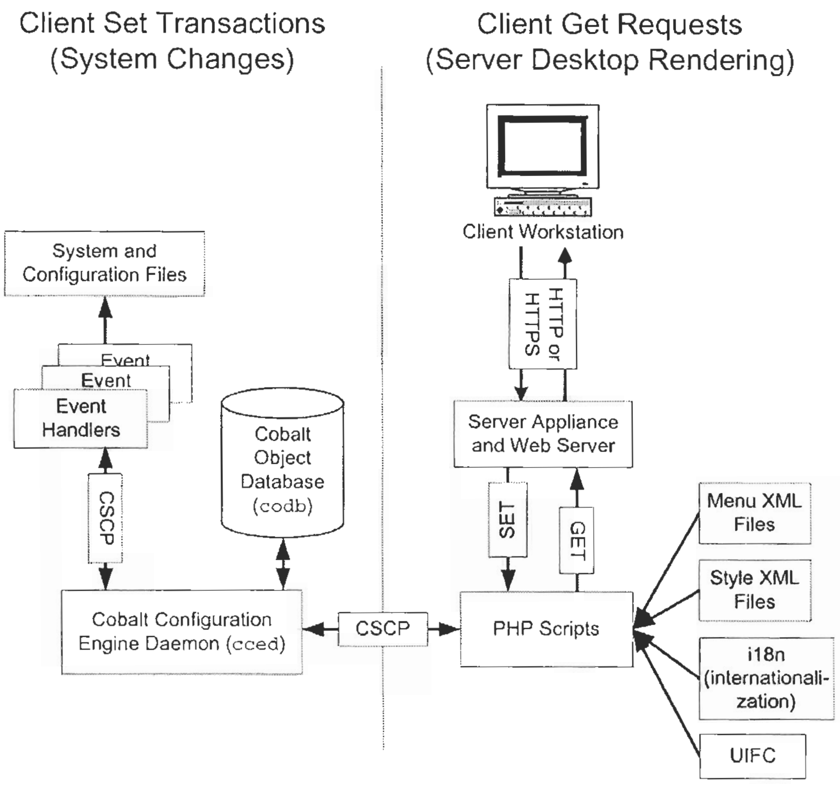

Client Set Transactions (System Changes)
Client Get Requests (Server Desktop Rendering)
Client Workstation
HTTP or HTTPS
Server Appliance and Web Server
SET
GET
Menu XML Files
Style XML Files
i18n (internationalization)
UIFC
PHP Scripts
CSCP
Cobalt Configuration Engine Daemon (cced)
Cobalt Object Database (codb)
CSCP
Event Handlers
Event
Event
System and Configuration Files

Figure 17-3 Sausalito components and process flow

Cobalt Configuration Engine

The main component of Sausalito is the Cobalt Configuration Engine (CCE), which is responsible for both SET transactions and GET requests. CCE is designed to act as a buffer between the system and the Server Desktop to keep the two logically separate.

CCE has several components:

- The Cobalt Object-oriented Database (codb)
- The Cobalt System Configuration Protocol (CSCP)
- Event handlers

CCE maintains the configuration state of Sausalito by:

- Initiating event handlers to change the state of the system
- Changing codb information
- Delivering information from codb to the Server Desktop

SET Transaction Overview

This section provides an overview of the SET transaction process and components and their functions within Sausalito.

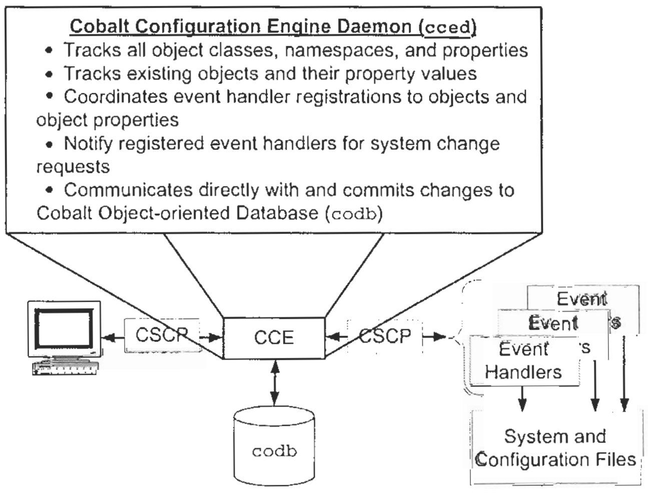
SET transactions are system changes performed by event handlers registered to objects or object properties being created, deleted, or modified. SET transactions are triggered by user input, such as:

- Adding a new object such as a user or group, or
- Modifying an existing object's properties, such as enabling or disabling a service or modifying user information

Then the event handlers registered to the object or the object properties are executed.

The CCE Daemon

The CCE daemon (cced) listens on the cced.socket, which is a standard UNIX socket, and handles incoming connections and signals from the CSCP. The CCE daemon is the administrator of Sausalito: It initiates event handlers based on change requests and commits changes to codb, when applicable. When you check the system processes on the server appliance, you should see the cced process running. Figure 17-4 shows the cced responsibilities:



Cobalt Configuration Engine Daemon (cced)

- Tracks all object classes, namespaces, and properties
- Tracks existing objects and their property values
- Coordinates event handler registrations to objects and object properties
- Notify registered event handlers for system change requests
- Communicates directly with and commits changes to Cobalt Object-oriented Database (codb)

Figure 17-4 The cced responsibilities

Cobalt Object-Oriented Database

In object-oriented programming, everything is an object. Objects have properties that contain values. For example, users and groups have names and disk quotas have sizes. In codb, the `System` object has properties for the server appliance's services. For example, the FTP service has a maximum-number-of-connections property, a maximum-number-of-megabytes-per-transfer property, and other properties.

Figure 17-5 shows a codb object expanded. Objects in codb are identified by numbers. Each object has a `CLASS` property that designates the type of the object (user, group, system, or network route, among others) and several other properties that contain configurable information. codb objects can be extended with additional properties.

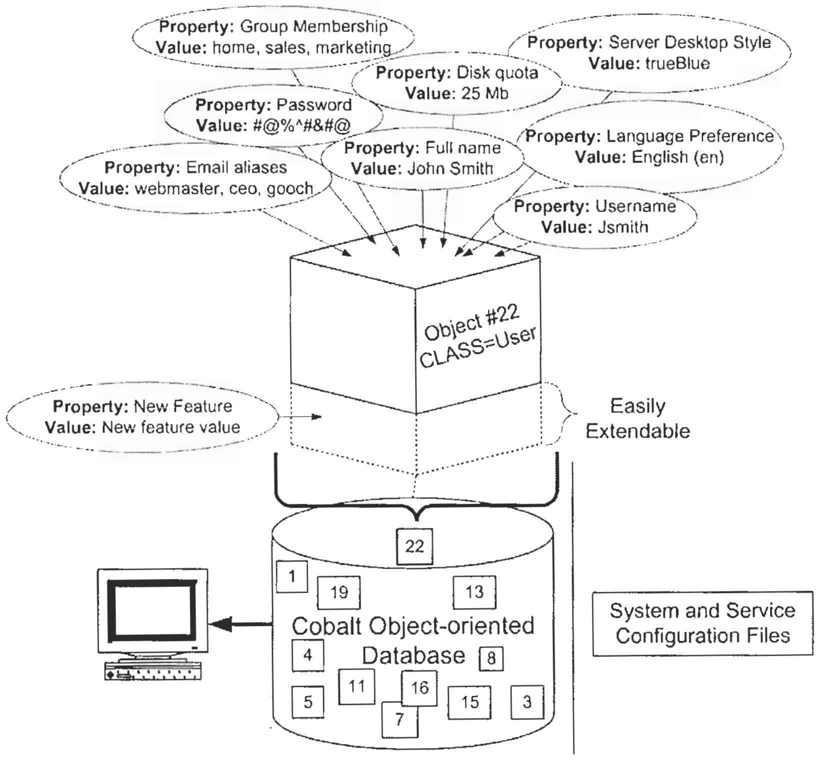


Figure 17-5 The codb with an expanded object

codb is a holding tank for objects and their property values. The information stored in codb is what is retrieved by CCE and viewed through the Server Desktop. For example, a user is an object and that user object has property values that include user name, full name, maximum disk space, email aliases, and other properties. These object property values display in the Server Desktop when you view a specific user's details.

Cobalt System Configuration Protocol

CSCP is a simple protocol used for communicating between clients and CCE, and between CCE and the event handlers. CSCP is the only mechanism by which CCE knows what actions to take. CSCP communicates requests from the client to CCE, and it communicates requests from CCE to the event handlers when system and service configuration files need to be added, removed, or modified. After the event handlers complete their work, either successfully or unsuccessfully, CSCP reports back to CCE with the status. Figure 17-6 shows the function of CSCP in Sausalito for handling the communication between requesting clients and CCE and between CCE and the event handlers.

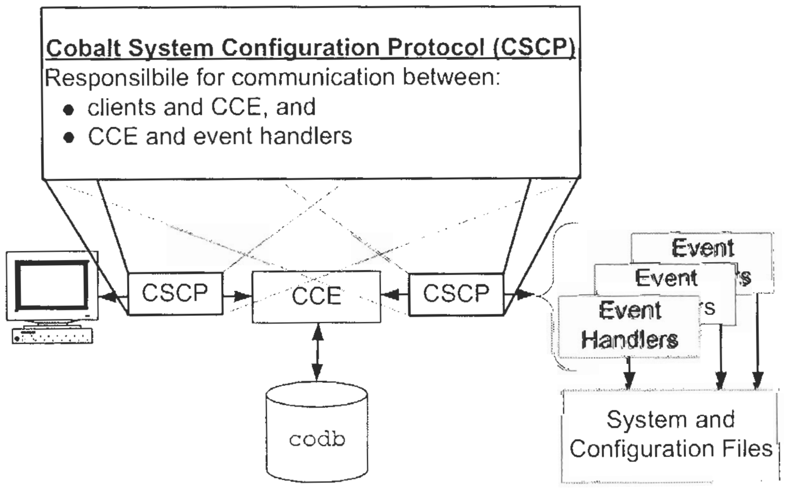

Cobalt System Configuration Protocol (CSCP)
Responsilbile for communication between:
• clients and CCE, and
• CCE and event handlers
CSCP
CCE
CSCP
codb
Event
Event
Event Handlers
System and Configuration Files

Figure 17-6 CSCP functions in Sausalito

Event Handlers

Event handlers are executable scripts that are triggered by CSCP. They have specific functions for changing system state, and registering services and applications. Services and applications register themselves in CCE by requesting notifications when specific codb objects and object property values are added, removed, or modified.

When these notifications take place, the event handlers make the changes to the system and report back, using CSCP, to CCE with the success or failure status.

Event handlers make it possible for software developers to apply changes to the system without affecting existing functionality, as long as guidelines are followed for adhering to the modularity of Sausalito. Figure 17-7 shows how the event handlers in Sausalito handle changes to the system and configuration files.

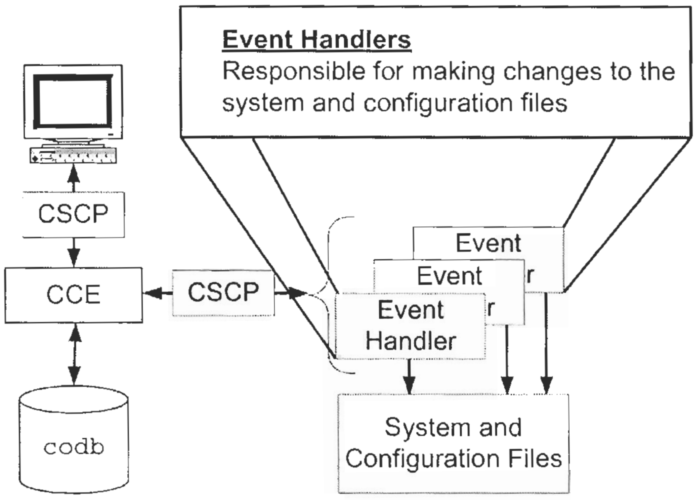
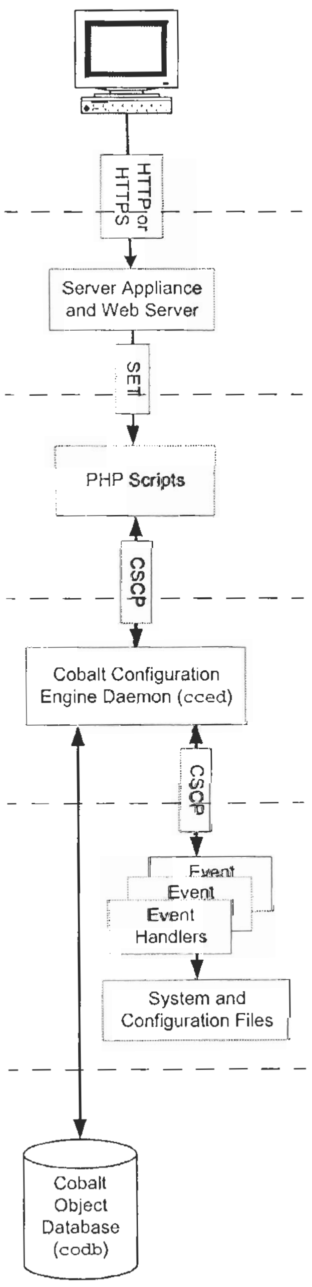

Event Handlers
Responsible for making changes to the system and configuration files
CSCP
CCE
CSCP
codb
Event
Event
Event Handler
System and Configuration Files

Figure 17-7 Event Handler functions in Sausalito

Table 17-1 on page 17-13 shows an overview of the process flow for SET transactions:

Table 17-1 SET transaction process


HTTP or HTTPS
Server Appliance and Web Server
SET
PHP Scripts
CSCP
Cobalt Configuration Engine Daemon (cced)
CSCP
Event
Event
Event Handlers
System and Configuration Files
Cobalt Object Database (codb)

A client makes a request to the Web server to create, destroy, or modify some aspect of the system, for example, adding a user or changing a service.

The Web server then sends the request to PHP scripts.

The PHP scripts communicate the request to the CCE daemon.

The CCE daemon triggers event handlers that have registered themselves to the object or object properties being created, destroyed, or modified.

The event handlers attempt, one at a time, to make changes to the system and configuration files, and then they report back to the CCE daemon with a successful or unsuccessful message. If an event handler is successful, the CCE daemon triggers the next event handler and the process repeats until there are no more event handlers. If an event handler is unsuccessful, the process is stopped and no changes are committed to the system.

If all the event handlers are successful, then the CCE daemon communicates directly with codb and commits the changes.

SET Transaction Configuration

Before SET transactions can take place in Sausalito, the system must be configured to recognize requested changes and system changes that must be made. This means:

- Objects and object properties need to be defined
- Event handlers need to be created and registered to the objects and object properties

Objects and object properties need to be defined for program packages, such as Apache, Proftpd, Active Monitor, Sendmail, and others. Then the objects and object properties need to be registered with the event handlers.

Note – When an object is added, deleted, or modified, the event handlers registered to that object or that object's properties are notified to make the necessary system changes.

Defining Objects and Object Properties

The structure of objects within `codb` is defined by `.schema` definition files. These definition files provide `schema`, `class`, `property`, and `typedef` information.

At system startup, the definition files are read; they define what objects can be created, the properties the objects can have, and the data type the object properties can contain. The schema definition files are located in the `/usr/sausalito/schemas/base/`*`service-type`* directories.

When `cced` is started, all files with a `.schema` file extension under the `/usr/sausalito/schemas` directory tree are used to build the object schemas.

The `/usr/sausalito/schemas/base/`*`service-name`* directories are strictly for organization. If you have installed third-party applications, you might also find `/usr/sausalito/schema/`*`vendor-name`* directories that contain schema definition files.

Table 17-2 contains the schema file elements and attributes discussed in this course. For a complete listing and description of the schema file elements and attributes, see the *Sun Cobalt Qube™ 3 Developer's Guide,* which is referenced in "Additional Resources" on page 17-2.

Table 17-2 Schema definition file elements and attributes

Element	**Attributes**	**Description**
`class`	`name`	Name of the class being defined or extended
	`version`	Version number for the class structure
	`namespace`	Optional name of the class structure
`property`	`name`	Name of the property
	`type`	Data type of the property
	`default`	Default value for the property
	`optional`	Defines whether the property can be an empty string("") or a valid fact used to make a decision
	`array`	Defines whether the property is an array of the specified type, can be any string, or is unspecified
`typedef`	`name`	Symbolic name for type
	`type`	Validation class for `typedef`
	`data`	Data validator: `re` = regular expression and `extern` = path to external program
	`errmsg`	Error message returned by CCE

Schema File Example

The following is an example of a fairly simple definition file that extends the `System` object with an FTP name space that has two properties: `enabled` and `maxconnections`. This file is written in XML. The first seven lines are comments that are opened by `<! --` and closed by `-->`.

```
1    <! --
2            $Id: ftp.schema,v 1.5 2000/09/19 23:48:28 thockin Exp $
3            Copyright(c) Cobalt Networks, Inc.
4            Author: asun@cobalt.com
5
6            This provides the schema for ftp.
7    -->
8
9    <class name="System" namespace="Ftp" version="1.0">
10           <property name="enabled" type="boolean" default="true"/>
11           <property name="maxConnections" type="int" default="25"/>
12   </class>
```

In line 9, `class name="System"` extends the `System` object with a new name space.

In line 10 and 11, the properties are defined for the `System.Ftp` object and name space. The `enabled` property has its `type` set to `boolean` and its `default` value set to `true`.

The `maxConnections` property has its `type` set to `integer` (`type="int"`) and its `default` value set to `25`.

Type Definitions

Type definitions define what type of data is allowed for a certain property. For example, `scalar` data is used for full user names; it allows first and last names to be any combination of letters, numbers, and white space.

If you have installed third-party software applications, you might find more type definitions defined in the `/usr/sausalito/schemas` directory. Table 17-3 on page 17-17 lists the server appliance type definitions (`typedef`), defined in the `/usr/sausalito/schemas/basetypes.schema` file.

Table 17-3 Server appliance type definitions and descriptions

Typedef	Description
`scalar`	Zero or more characters of any value
`word`	One or more non-whitespace characters
`alphanum`	One or more alphanumeric characters
`int`	A positive integer; any integer except zero preceded by an optional unary minus (negative) sign, or zero
`uint`	An unsigned integer, which can be any positive integer greater than or equal to zero
`boolean`	A logical TRUE or FALSE; an empty string (""). Or, the value zero, which is a logical FALSE, and any other value, which is a logical TRUE
`ipaddr`	A set of four integers between 0 and 255 separated by periods, for example, 192.168.1.254
`network`	A network number, such as 10.9.0.0/16
`email_address`	The address of an email user; for example, `harry@suncobalt.com`
`netmask`	A number from 1 to 32, or a dot-quadded IP mask
`fdqn`	A fully qualified domain name; for example, `www.suncobalt.com`
`hostname`	A non-period host name; for example, www or `smtp`
`domainname`	A fully qualified domain name or a domain name; for example, `www.suncobalt.com` or `suncobalt.com`
`password`	Scalar data, masked in cced logs

Exercise: Extending an Object's Schema

In this exercise you add functionality to your FTP server. The feature you add and make available to the admin user through the Server Desktop is root user access feature. This feature is disabled by default by the Proftpd application. Extending the schema is the first part of the process to add this feature to the Server Desktop. You extend the schema of the `System` object by adding a `FtpRootLogin` name space with an `enabled` property.

Tasks

Complete the following steps to set up a directory structure for your FTP feature:

1. Open a telnet window and change to the `/usr/sausalito/schemas` directory, create new directory using *`your_first_name/ftp`* and then change to that directory:

   ```
   # cd /usr/sausalito/schemas
   # mkdir -p your_first_name/ftp
   # cd your_first_name/ftp
   ```

2. Copy the `ftp.schema` file from the server-appliance-specific directory to your current directory and rename the file to `ftproot.schema`:

```
# cp /usr/sausalito/schemas/base/ftp/ftp.schema ftproot.schema
```

Complete the following steps to create and edit your new schema file:

1. Open the `ftproot.schema` file editing and switch to insert mode:

   ```
   # vi ftproot.schema
   i
   ```

2. On the line that starts with `<class name="System"`, change `Ftp` to **`FtpRootLogin`**.
3. On the line that starts with `<property name="enabled"`, change `true` to `0`.
4. Switch to command mode by pressing the Escape key.

5. Move the cursor down one line and remove the line:

   ```
   dd
   ```

   Your final results should look like this, after the initial file comments:

```
<class name="System" namespace="FtpRootLogin" version="1.0">
      <property name="enabled" type="boolean" default="0"/>
</class>
```

6. Save and close the file.

   ```
   :wq
   ```

Complete the following steps to restart the CCE server, making the new name space and property available. Then, check for errors:

1. Check for errors by restarting CCE and then make sure CCE restarted:

   ```
   # /etc/rc.d/init.d/cced.init restart
   Shutting down cced: done
   Starting cced:   cced
   Running CCE constructors:
   [root ftp]# ps -e | grep cced
   8584 ?          00:00:00 cced
   [root ftp]#
   ```

   If you do not receive a similar result as above, make sure the syntax in your `ftproot.schema` file is correct.

Registering an Event Handler

To register an event handler with the system, create an event handler configuration file with a `.conf` extension. The `.conf` file must contain the information that defines the class, name space, and properties for the object to which you want the event handler to be registered. Then, if the object is modified, the event handler will be triggered and make the necessary changes to the operating system. The `.conf` file must also contain the path to the executable script that is the event handler itself.

The `.conf` file can be placed anywhere under the `/usr/sausalito/conf` directory tree. This makes it possible for third-party software developers to create their own `/usr/sausalito/conf/`*`vendor-name`* directories, in which they can store their `.conf` files.

CCE finds all the `.conf` files in the `/usr/sausalito/conf` directory and subdirectories; however, all file names that begin with a period are ignored.

Event Handler File Format

The `.conf` file format is simple and can contain several lines with three fields per line. For example:

Class.Event   Handler definition(File path)   Execution stage

Class.Event

Class.Event is a string of text that defines:

- An object
- An optional name space
- A property or an object
- An optional name space (optional)
- An event based on the events listed in Table 17-4

Table 17-4 Valid class events

Event	**Definition**
`_CREATE`	When an object of the specified class is created

Table 17-4 Valid class events (Continued)

Event	Definition
_DESTROY	When an object of the specified class is destroyed
propertyname	When the specified property of the specified class is changed
*	Wildcard, which means when any property of the specified class or `class.namespace` is changed

Handler Definition

The handler definition is the path to the event handler's executable script. This path is preceded by either `perl:` or `exec:`. The `perl:` text causes CCE to use a persistent Perl daemon, thereby expediting the execution request. The `exec:` text executes the script or binary.

Execution Stage

The execution stage allows the handler writer to suggest a relative order for event handler execution. The following five defined, case-sensitive execution stages are listed in priority from highest to lowest:

- `VALIDATE`
- `CONFIGURE`
- `EXECUTE`
- `TEST`
- `CLEANUP`

If the execution stage is not specified, the event handler is in the `EXECUTE` stage. Relative ordering between stages is guaranteed, but ordering within a given stage is not.

Example: Event Handler Configuration File

The following `/usr/sausalito/conf/ftp/ftp.conf` file is provided as an example. When any of the properties are modified on the `System.Ftp.*` or `System.FileShare.*` object and name space, the `/usr/sausalito/handlers/base/ftp/system.pl` script is executed to handle the request.

This file also controls the script for enabling anonymous FTP, and for enabling and disabling FTP service monitoring using the Active Monitor.

```
# $Id: ftp.conf,v 1.10 2000/09/15 01:03:54 thockin Exp $
# handlers for ftp
#
System.FileShare.*      perl:/usr/sausalito/handlers/base/ftp/guest.pl
System.Ftp.*            perl:/usr/sausalito/handlers/base/ftp/system.pl
ActiveMonitor.FTP.enabled        perl:base/am/am_enabled.pl EXECUTE
ActiveMonitor.FTP.monitor        perl:base/am/am_enabled.pl EXECUTE
System.Ftp.enabled      perl:base/ftp/enableAM.pl
```

Exercise: Creating and Registering an Event Handler

In this exercise you continue the process started in "Exercise: Extending an Object's Schema" on page 17-18. There, you extended the properties of the FTP server by adding the FTP root user access feature though the Server Desktop. Now complete the tasks of creating the event handler configuration file and writing the executable script for modifying the FTP server configuration file.

Tasks

Complete the following steps to create the event handler configuration file:

1. Open a telnet window and change to the `/usr/sausalito/conf` directory, create new directory using *`your_first_name/ftp`* and then change to that directory:

   ```
   # cd /usr/sausalito/conf
   # mkdir -p your_first_name/ftp
   # cd your_first_name/ftp
   ```

2. Open for editing a `ftpRootAccess.conf` file and switch to insert mode:

   ```
   # vi ftpRootLogin.conf
   i
   ```

3. Add the following text (use a tab for the space between `enabled` and `perl`):

```
# handler for ftp root login
#
System.FtpRootLogin.enabled         perl:/usr/sausalito/handlers/your_first
_name/ftp/rootLogin.pl
```

   Make sure you replaced the *your_first_name*, with your real first name!

4. Switch to command mode by pressing the Escape key.
5. Save and close the file:

   ```
   :wq
   ```

Complete the following steps to write the executable event handler Perl script for adding the root access directive to the FTP server configuration file:

1. Change to the /usr/sausalito/handlers directory, create new directory using *your_first_name/ftp* and then change to that directory:

   ```
   # cd /usr/sausalito/handlers
   # mkdir -p your_first_name/ftp
   # cd your_first_name/ftp
   ```

2. Open for editing a rootLogin.pl file and switch to insert mode:

   ```
   # vi rootLogin.pl
   i
   ```

3. Insert the following text exactly as shown into the file. Use the Tab key to indent lines. It might take several attempts to get the text exactly right, so take your time.

Note – This script is written to be streamlined and it does not follow good programming syntax guidelines for providing comments about the script's use.

```
#!/usr/bin/perl -w -I/usr/sausalito/perl -I.
# System.FtpRootLogin.enabled modify handler
use strict;
use Sauce::Config;
use Sauce::Util;
use CCE;

my $cce = new CCE;
$cce->connectfd();

my $oid = $cce->event_oid();
my($ok, $obj) = $cce->get($oid, 'FtpRootLogin');

if (!$oid) {
	$cce->bye('FAIL');
	exit(1);
}

#
my $enabled = $obj->{enabled} ? "on" : "off";

my $fun = sub {
	my ($fin, $fout) = (shift, shift);
```

```
        my $found = 0;
        my $conf = "RootLogin\t\t\t$enabled\n";

        while (defined($_ = <$fin>)) {
                if(/^\s*RootLogin/) {
                        print $fout $conf;
                        $found = 1;
                } else {
                        print $fout $_;
                }
        }
        print $fout $conf unless ($found);
        return 1;
};

my $ret = Sauce::Util::editfile("/etc/proftpd.conf",
$fun);

if ($ret) {
        $cce->bye('SUCCESS');
        exit(0);
} else {
        $cce->bye('FAIL');
        exit(1);
}
```

4. Switch to command mode by pressing the Escape key.
5. Save and close the file:

   ```
   :wq
   ```

6. Execute the file at the command line to test the syntax:

   ```
   # perl -cw rootLogin.pl
   rootAccess.pl syntax OK
   [root ftp]#
   ```

   The -cw option tells Perl to compile, but not execute, the program and to display warnings about errors.

   If you receive warnings, check the contents of the `rootAccess.pl` file and make any necessary changes until you receive no more warnings.

7. Restart CCE:

```
# /etc/rc.d/init.d/cced.init restart
Shutting down cced: done
Starting cced:  cced
Running CCE constructors:
[root ftp]#
```

CSCP Communication With CCE

Communication to CCE is done through CSCP and there are two modes of communication: client mode and handler mode. The handler mode has more commands than the client mode.

CCE Communication Modes

CSCP is shown in Table 17-1 on page 17-13. CSCP is used in the *handler mode* for communication between CCE and the event handlers; it is also used for communication between CCE and the Server Desktop.

CSCP is used in the *client mode* to establish a direct connection to CCE. To establish a direct connection to CCE, the `cceclient` utility is executed from the command line: `/usr/sausalito/bin/cceclient`. Table 17-5 lists the commands used for both client and handler CCSP communication modes.

Table 17-5 The client and handler mode commands

Command	Description
`auth`	Authenticates as a user to get that user's access privileges (starts a new session)
`authkey`	Authenticates to an already existing session
`endkey`	Expires the current `sessionid`
`whoami`	Returns the `oid` (Object Identification) of the currently authenticated user
`bye`	Closes the connection
`commit`	Triggers any postponed handler activity
`create`	Creates a new object of a certain class
`destroy`	Destroys an object
`find`	Finds all objects that match a given criteria
`get`	Gets all properties for a certain object or object name space
`names`	Lists name spaces associated with a class
`classes`	Lists all classes
`set`	Sets the properties of a certain object

Table 17-6 contains additional commands that are available in the handler mode when the CCE is communicating with an event handler:

Table 17-6 Additional Handler Mode Commands

Command	**Description**
`baddata`	Reports that an unrecognized attribute or value was passed
`info`	Reports a piece of information
`warn`	Reports a warning or error

CCE Response Codes

For every command given to CCE, a response code is returned. These response codes consist of a numeric code and a set of arguments. The value of the first digit of the numeric code defines whether the message is an informational message (1), a success message (2), a warning message (3), a failure message (4), or a system-issued message (9). Table 17-7 describes the response types.

Table 17-7 CCE Response Codes

Numeric Code Range	**Type of Response**
100-199	Informational
200-299	Success
300-399	Warning
400-499	Failure
900-999	System-issued message (which can be sent at any time)

Exercise: Testing Systems Changes From `cceclient`

In this exercise you test the new feature added to the FTP server for root login access. This test must be successful before you extend the Server Desktop with new menu items.

Tasks

Complete the following steps to determine the object number for the `System` object and test the FTP server changes performed by the event handler:

1. In telnet, communicate to CCE using the `cceclient` utility and fine the number for the `System` object:

   ```
   # /usr/sausalito/bin/cceclient
   100 CSCP/0.80
   200 READY
   find System
   104 OBJECT 1
   201 OK
   ```

2. Open another telnet session and list the `/etc/proftpd.conf` file contents:

   ```
   # cat /etc/proftpd.conf
   ```

   Notice that no `RootLogin` directive is listed in the file.

3. In the first telnet session that has `cceclient` running, for the `System` object (object `1`), set the `FtpRootLogin` name space's `enabled` property to `0`. (Zero equals off.)

   ```
   set 1.FtpRootLogin enabled="0"
   201 OK
   ```

4. In the other telnet session, list the `/etc/proftpd.conf` file:

   ```
   # cat /etc/proftpd.conf
   ```

   Notice the last line of the file reads `RootLogin          off`

5. Try to log in to the FTP server as root and exit:

   ```
   # ftp localhost
   Connected to localhost.
   220 ProFTPD 1.2.0 Server (ProFTPD)
   [www.server_appliance.com]
   Name (localhost:admin): root
   331 Password required for root.
   Password: your_password
   530 Login incorrect.
   Login failed.
   ftp> exit
   ```

   Notice the `Login Failed.`

6. In the telnet window that has `cceclient` running, set the `System` object's `FtpRootLogin` name space's `enabled` property to `1`. (One equals on.)

   ```
   set 1.FtpRootLogin enabled="1"
   201 OK
   ```

7. In the other telnet session, list the `/etc/proftpd.conf` file:

   ```
   # cat /etc/proftpd.conf
   ```

   Notice that the last line of the file reads `RootLogin` on

8. Log in to the FTP server as root and exit:

   ```
   # ftp localhost
   Connected to localhost.
   220 ProFTPD 1.2.0 Server (ProFTPD)
   [www.server_appliance.com]
   Name (localhost:admin): root
   331 Password required for root.
   Password: your_password
   230 User root logged in.
   Remote system type is UNIX.
   Using binary mode to transfer files.
   ftp> exit
   ```

   Notice the `230 User root logged in.`

GET Requests

GET transactions deliver Hypertext Markup Language (HTML) content to the end-user's browser. This is done using a combination I18n (internationalization) components, PHP scripting, and XML menu item and style files.

The Server Desktop (Figure 17-8) in the server appliance is a dynamic structure. It is generated from a set of files that define its menu items, language, and style. Whenever a user logs in to the Server Desktop, these files are examined and the Server Desktop is generated.

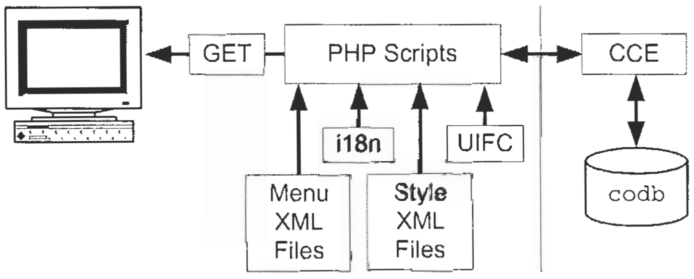


Figure 17-8 Sausalito Server Desktop rendering

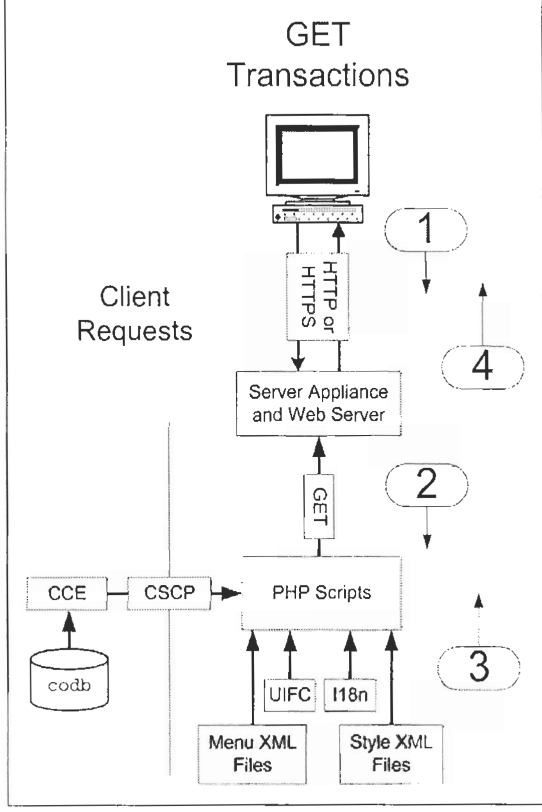
To explain the components and processes involved in generating the Server Desktop, the remaining sections of this module examine the following items.

- Server Desktop screen-generation process
- Server Desktop configuration file
- Server Desktop menu structure
- I18n components
- User Interface Foundation Classes (UIFC)
- HTML Component Factory
- Style definition files

Server Desktop Screen-Generation Process

To display data and images on the monitor, the client browser sends a request to the Apache Web server with PHP compiled in. The Apache Web server executes PHP scripts, reads the Server Desktop configuration file, retrieves object information and property values from codb, retrieves menu structure and style information from XML files, retrieves language preferences from the browser or user object, and retrieves the Server Desktop elements from the UIFC.

The Apache Web server then creates HTML code and sends it to the client browser to render the Server Desktop. Figure 17-9 describes the Server Desktop screen-generation process.



1. The client requests the administration pages by logging in. The request is handled by the Web server.
2. The Web server in turn passes the request to PHP scripts.
3. The PHP scripts gather the information to form the Server Desktop from:
   - CCE (for the users, groups, system, and services status)
   - I18n (to pull the correct language text for menu items and help text)
   - XML files (for the style and the menu structure)
   - UIFC (for the Server Desktop widget information and behavior)
4. After everything is pulled together by the PHP scripts, it is sent back to the client as HTML code.

Figure 17-9 The screen-generation process

Server Desktop Configuration File

The PHP scripts determine the file-system locations for the menu and style definitions by reading the `/usr/sausalito/ui/conf/ui.cfg` Server Desktop configuration file. This file defines locations for the Server Desktop screen generation.

Navigation Sub-System Overview

The navigation system within the Server Desktop manages the site maps. Users can modify site maps by adding and removing nodes, which are then interpreted by navigation managers.

Server Desktop Rendering

The structure and contents, known as the site map, of the Server Desktop are generated from XML files that are simple and contain one menu item, also known as nodes, per file. These XML files are located under the `/usr/sausalito/ui/menu/` directory and subdirectories.

To add a node to the site map, create a new XML file under the `/usr/sausalito/ui/menu/` directory. To remove a node from the site map, remove the node's XML file. The names of the XML files and the subdirectories in the `/usr/sausalito/ui/menu/` directory have no impact on the site map definition. The subdirectories are merely used for organizing the XML site map node files.

For organizational purposes, the Server Desktop XML menu item files are located in *`service-name`* and *`function-name`* subdirectories (for example, `addressbook/`, `maillist/`, and `backup/`) under the `/usr/sausalito/ui/menu/base` directory.

Third-party software developers can create new directories under the `/usr/sausalito/ui/menu` directory to keep their menu item files organizationally separate.

Menu Item File Examples

Figure 17-10 shows an expanded FTP Menu. You can access this menu from the Server Desktop by clicking Administration tab ‰ File & Print ‰ FTP menu. The callouts on Figure 17-10 refer to the text below the figure. These are the actual XML files that define the Administration tab, the File & Print menu group, and the FTP menu item (or site map nodes).

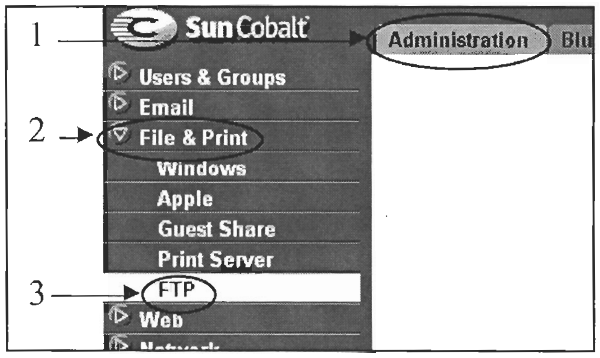


Figure 17-10 Server Desktop, File Services ‰ FTP Menu expanded

1. Administration tab file: `administration.xml` (these code lines wrap)

```
<item id="base_administration"
        label="[[base-carmel.base-administration]]"
        description="[[base-carmel.base-administration_description]]"
        url="/splashAdmin.php"
        requiresChildren="1">
  <parent id="root" order="10">
    <access require="systemAdministrator"/>
  </parent>
</item>
```

2. File Services menu group file: `fileSharing.xml`

```
<item id="base_fileSharing" label="[[base-fileshare.fileSharing]]"
description="[[base-fileshare.fileSharing_help]]" >
  <parent id="base_administration" order="30"/>
</item>
```

3. FTP menu item file: `ftp.xml`

```
<item id="base_ftp" label="[[base-ftp.ftp]]" description="[[base-
ftp.ftp_help]]" url="/base/ftp/ftp.php">
  <parent id="base_fileSharing" order="50" >
        <access require="adminFtpServer"/>
  </parent>
</item>
```

Menu Item File Attributes

The `administration.xml` file contains the text `<access require="systemAdministrator"/>`. This text indicates that, for a user to access the Administration tab, the user's `uiRights` property must be set to `systemAdministrator`.

Also, to gain access to any XML menu file that uses the `administration.xml` file's `base_administration` item ID, the parent menu item file also has to meet the same `uiRights` property requirement.

Table 17-8 XML file elements and attributes

Element	Attribute	Description
`item`	`id`	The `id` item must be unique among the XML files. Therefore, it is advisable to prepend a vendor tag to the `id`.
	`label`	The `label` item is a short readable string that labels the node. Navigation managers can display a list of labels for users to navigate to.
	`description`	A `label` can sometimes be too short. A `description` is used complement the `label` by describing what the node is about.
	`type`	The `type` item is used by navigation managers to distinguish items. Navigation managers can then act on the items differently.
	`url`	The `url` points to the content page of this node.

Table 17-8 XML file elements and attributes (Continued)

Element	Attribute	Description
`parent`	`id`	This is the `id` of the parent node, as described in the `item id` element. Nodes can have multiple parents.
	`order`	When there are several children nodes under a parent node, the navigation managers might need to know which child to use first. The smaller the `order` integer, the more important the node.
	`access require`	This attribute is used to specify the access rights for the resource (the XML menu item file). If the value of the user object's uiRights property does not equal the value of this attribute, the user can not view the menu item, and therefore the user can not administrate or modify the system for that resource.

Parent Access Attribute

In the ftp.xml file listing, there is a line that says `<access require="adminFtpServer">`. Access to the parent elements is controlled by this `access require` element:

- If there is no `access require` element, the parent link has no access control and anyone can traverse the link.
- If there is one `access require` element, access is granted if and only if this requirement is met.
- If there is more than one `access require` element, access is granted if any one of the multiple requirements is met. In other words, this is an "or" condition.

You learn about the `access require` element in "Defining Administration Capabilities for Users" on page 17-48.

Exercise: Extending Server Desktop Menu Items

In this exercise you create the necessary XML files to add a menu item to the Server Desktop for your application.

Tasks

Complete the following steps to create the `ftp.xml` file:

1. In a telnet window navigate to the `/usr/sausalito/ui/menu/` directory, create a new directory using *`your_first_name/service-name`* and change into that directory:

   ```
   # cd /usr/sausalito/ui/menu/
   # mkdir -p your_first_name/ftp
   # cd your_first_name/ftp
   ```

2. Copy the `/usr/sausalito/ui/menu/base/ftp/ftp.xml` file into your new `ftp` directory and open for editing:

   ```
   # cp /usr/sausalito/ui/menu/base/ftp/ftp.xml ftp.xml
   # vi ftp.xml
   ```

3. Use the following three commands to make text changes in the file:

   ```
   :s/base/your_first_name/g
   :%s/50/60/g
   :%s/Server/RootLogin/g
   ```

   Leave the `parent id="base_fileSharing"` text alone, because you are building off the existing server appliance menu navigation item.

4. Save and close the file:

   ```
   :wq
   ```

Complete the following steps to view your menu item:

1. Open your browser and log in as admin to the Server Desktop. If you are logged into your Server Desktop already, log out and then log back in.

2. Click File & Print.

   Your results should similar to Figure 17-11.

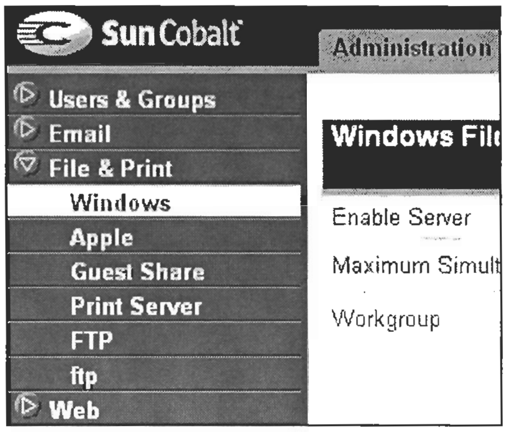


Figure 17-11 Server Desktop added menu item

Internationalization

Sausalito has integrated internationalization. (Internationalization is often referred to as "I18n," which stands for "I - eighteen letters - n.") I18n is the process of planning and implementing products and services so that they can easily be adapted to different languages.

Localization is also part of the I18n component. ("Localization" is often referred to as "L10n," which stands for "L - ten letters - n.") L10n is the process of adapting a product or service to a particular language, culture, and look and feel.

Usually, programs are written and documented in English, and they use English at execution time to interact with users. This is not the case with Sausalito on the server appliance platform. The I18n implementation in Sausalito is based on the GNU `gettext` tools, and it is designed to minimize the impact of I18n.

I18n Components

The I18n directories are located under the `/usr/share/locale/` directory. A listing of the `/usr/share/locale/` shows the locale directories, whose names are equal to the ISO-636 language codes and ISO-3166 country codes.

By using the ISO codes, languages are packaged as components. L10n components contain only locale-sensitive data that specifies a particular locale. For example, strings, currency format, date format, and number format are packaged into locale components. Strings are stored in *`domain`*`.mo` files under `/usr/share/locale/`*`ISO code`*`/LC_MESSAGES/` directories.

Selecting a Language

By logging in and navigating to Personal Profile ‰ Account ‰ Account Settings, users can change their language preference from the Language Preference drop-down list.

By default, when users are created, their language preference is set to <*Set From Browser Options*>, which means the server appliance determines which language to use by the language to which the user's browser is set.

If the language to which the browser is set is not available on the server appliance, the server appliance uses the default language setting defined in the `/usr/sausalito/ui/conf/ui.cfg` file. By default, this is set to `defaultLocale=en` (English).

In Sausalito, which language to use for the Server Desktop is based on the two-letter ISO-639 language codes, the two-letter ISO-3166 country codes, and optional variant codes. Figure 17-12 shows the Sausalito language selection process.

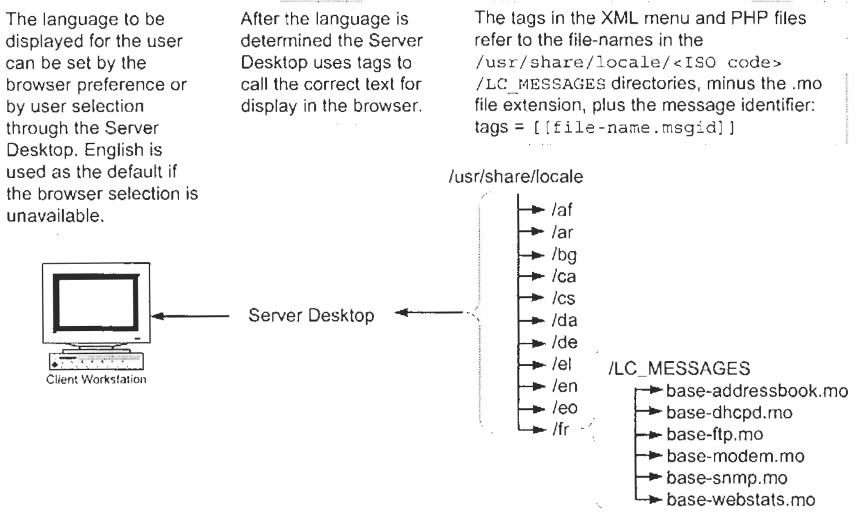

The language to be displayed for the user can be set by the browser preference or by user selection through the Server Desktop. English is used as the default if the browser selection is unavailable.
After the language is determined the Server Desktop uses tags to call the correct text for display in the browser.
The tags in the XML menu and PHP files refer to the file-names in the /usr/share/locale/<ISO code>/LC_MESSAGES directories, minus the .mo file extension, plus the message identifier: tags = [[file-name.msgid]]
/usr/share/locale
/af
/ar
/bg
/ca
/cs
/da
/de
/el
/en
/eo
/fr
/LC_MESSAGES
base-addressbook.mo
base-dhcpd.mo
base-ftp.mo
base-modem.mo
base-snmp.mo
base-webstats.mo
Server Desktop
Client Workstation

Figure 17-12 Server Desktop language selection process

Adding New Languages

Adding new languages is straightforward. The difficulty is keeping track of all the message IDs available through the Server Desktop and where they are all placed. Figure 17-13 shows the process for making I18n components available for Sausalito GET transactions.

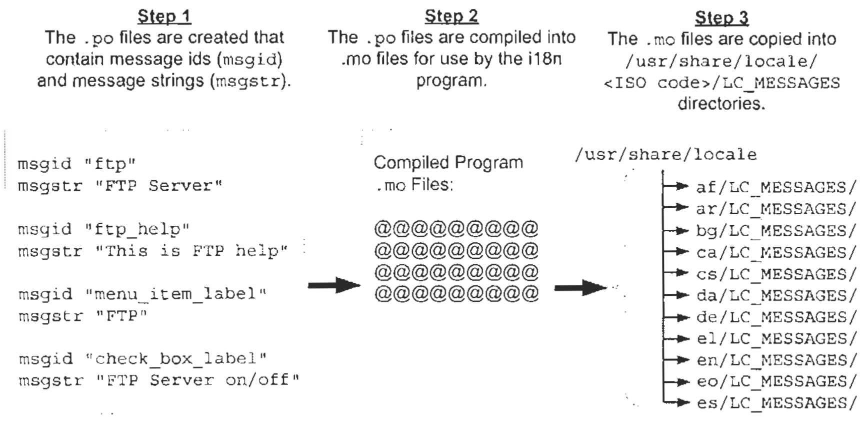


Figure 17-13 I18n language availability process

Exercise: Adding an I18n Server Desktop Message File

In this exercise you create text for your new feature in the Server Desktop that uses the I18n component for language expandability.

Tasks

Complete the following steps to create a vendor-specific .po file:

1. Navigate to the /opt directory, create a new directory using *your_first_name*/i18n/po-files/en and then change to that directory:

```
# cd /opt
# mkdir -p your_first_name/i18n/po-files/en
# cd your_first_name/i18n/po-files/en
```

2. Open for editing a *your_first_name*-ftp.po file:

```
# vi your_first_name-ftp.po
i
```

3. Enter the following message identifiers (msgid) and strings (msgstr), starting the file with one blank line on top:

[blank line, no text]

```
msgid "ftp"
msgstr "FTP Root Login"

msgid "ftp_help"
msgstr ""
"Turn the FTP root user login on or off, by enabling "
"the above check box, and saving the changes"

msgid "nameTagHelp"
msgstr "Can manage FTP root login feature"

msgid "nameTag"
msgstr "Manage FTP Root Login"

msgid "ftpSettings"
msgstr "FTP Server Root User Login Setting"

msgid "enableRootLogin"
msgstr "Enable Root User Login"

msgid "enableRootLogin_help"
msgstr "[[jeff-ftp.ftp_help]]"
```

4. Save and close the file:

```
:wq
```

Complete the following steps to compile the .po file into the .mo file, copy to the I18n directory, and restart the Server Desktop server to make the new text available for use.

1. Compile the *your_first_name*-ftp.po file into the English (en) I18n component directory as *your_first_name*-ftp.mo:

```
# msgfmt -e your_first_name-ftp.po -o
/usr/share/locale/en/LC_MESSAGES/your_first_name-ftp.mo
```

2. Restart the Server Desktop server (admserv):

```
# /etc/rc.d/init.d/admserv restart
Stopping admin web server: ahttpd
Starting admin web server: ahttpd
[root po-files]#
```

   In upcoming labs, the correctness of your I18n component programming is checked.

User Interface Foundation Classes

The User Interface Foundation Classes (UIFC) is a comprehensive set of class libraries for the Server Desktop components. The UIFC is made from the PHP and JavaScript™ files located in the `/usr/sausalito/ui/libPhp/uifc` and `/usr/sausalito/ui/web/libJs/uifc` directories, respectively.

These directories contain files that generate HTML code for page rendering and JavaScript code for error checking. The `/usr/sausalito/ui/libPhp/uifc` directory contains the `*.php` files used for HTML code generation to build the Server Desktop elements. The `/usr/sausalito/ui/web/libJs/uifc` directory contains the `*.js` files used for data-entry error checking.

The UIFC make object classes available to software developers to simplify Server Desktop creation. These class libraries contain the templates for building visual components, such as buttons, bars, Boolean components, form fields, domain names, email addresses, IP addresses, scroll lists, radio buttons, time zone formats, and so on. The UIFC are designed for seamless integration with I18n.

HTML Component Factory

The HTML Component Factory simplifies the task of instantiating and initializing the Server Desktop elements. This factory is a template file for what you see in the Server Desktop, and it handles most cases of instantiating and initializing objects.

Note – To use the UIFC, you need to understand object-oriented design and programming, as well as PHP. This is because the UIFC classes are object-oriented and implemented in PHP. Detailed information about the use of the UIFC with PHP programming is beyond the scope of this course.

Exercise: Building Pages Using the UIFC and I18n

In this exercise you build Server Desktop pages for your FTP root access feature. You modify existing PHP code and test its functionality as follows:

Tasks

Complete the following steps to create a Server Desktop page file for your new FTP server feature:

1. In telnet, navigate to the `/usr/sausalito/ui/web` directory and create a new directory using your first name and then change to that directory.

   ```
   # cd /usr/sausalito/ui/web
   # mkdir -p your_first_name/ftp
   # cd your_first_name/ftp
   ```

2. Copy the `/usr/sausalito/ui/web/base/ftp/ftp.php` file into your new `ftp` directory and open the file for editing:

   ```
   # cp /usr/sausalito/ui/web/base/ftp/ftp.php ftp.php
   # vi ftp.php
   ```

3. Enter the following three commands to change text in the file:

   ```
   :%s/base/your_first_name/g
   :%s/"Ftp"/"FtpRootLogin"/g
   :%s/ServerField/RootLogin/g
   ```

4. Remove the following four lines from the file:

```
$block->addFormField(
  $factory->getInteger("maxUserField", $ftp["maxConnections"], 1, 1024),
  $factory->getLabel("maxUserField")
);
```

5. Save and close the file:

   ```
   :wq
   ```

Complete the following steps to create the Server Desktop page handler file for your new feature:

1. Copy the `/usr/sausalito/ui/web/base/ftp/ftpHandler.php` file into your `ftp` directory and open the file for editing:

   ```
   # cp /usr/sausalito/ui/web/base/ftp/ftpHandler.php
   ftpHandler.php
   # vi ftpHandler.php
   ```

2. Enter the following three commands to change text in the file:

   ```
   :%s/ServerField,/RootLogin);/g
   :%s/"Ftp"/"FtpRootLogin"/g     (Make sure you use capital 'F')
   :%s/base/your_first_name/g
   ```

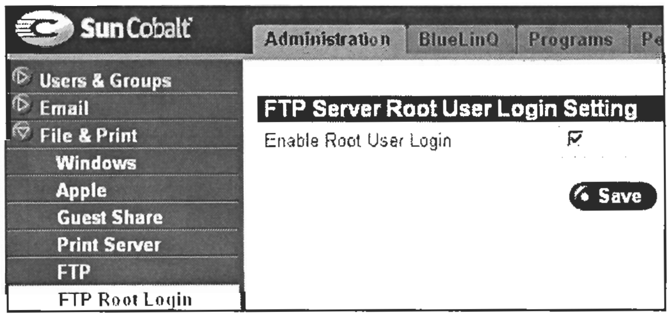
3. Remove the line, which reads `"maxConnections" ...`, then save and exit:

   ```
   /max
   dd
   :wq
   ```

Complete the following steps to view your page in the Server Desktop:

1. Open your browser log out and back in as admin.
2. Click File & Print.
3. Click FTP Root Access.

   Your results should look similar to :



FTP Server Root User Login Setting

Enable Root User Login ✓

Figure 17-14 Successful I18n Server Desktop text addition

If you do not receive the results as above, go back and verify that your file syntax is correct, and repeat the process until you receive the results shown in Figure 17-14.

Complete the following steps to test the FTP root access feature:

1. In telnet, view the last line of the `/etc/proftpd.conf` file:

```
# tail -1 /etc/proftpd.conf
RootLogin               on
```

2. In your browser, uncheck the Enable Root User Access feature, and click Save.

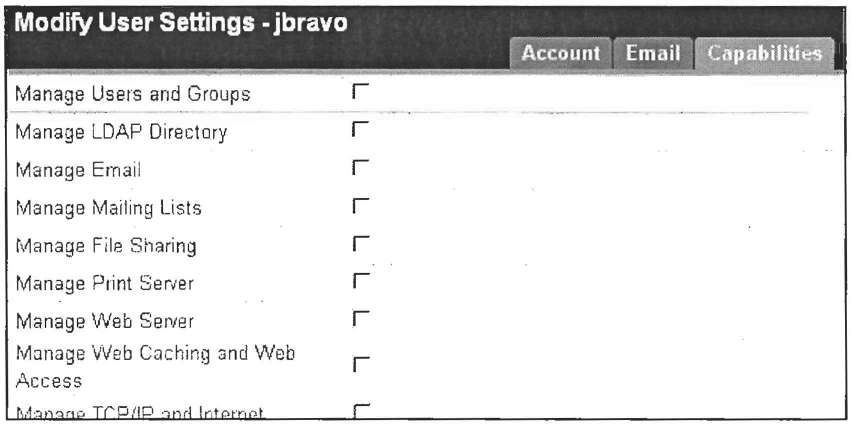
3. In telnet, view the last line of the `/etc/proftpd.conf` file:

```
# tail -1 /etc/proftpd.conf
RootLogin               off
```

Defining Administration Capabilities for Users

One feature on the server appliance is the ability to distribute the administration tasks, known as capabilities, to regular users. Distributing capabilities to regular users is handled within the Sausalito.

Capabilities are made available in the Server Desktop by creating a capability object. When the admin accesses a users Administration tab > Users and Groups ‰ User List ‰ Modify User ‰ Capabilities tab, all capability objects are displayed as shown in Figure 17-15.



Modify User Settings - jbravo

Manage Users and Groups	☐
Manage LDAP Directory	☐
Manage Email	☐
Manage Mailing Lists	☐
Manage File Sharing	☐
Manage Print Server	☐
Manage Web Server	☐
Manage Web Caching and Web Access	☐
Manage TCP/IP and Internet	☐

Figure 17-15 User capability settings

Once the admin has selected the capabilities for a user, the access to administrator the capability is handled in Sausalito. What happens is the user allowed to access the administration screens associated with the capability. The user does not actually receive admin permissions to the service, but is allowed access through the Server Desktop to modify the feature or service.

If a regular user is giving the capability to Manage Users and Groups, the user can pass along the capability other users, for which the user has access. However, the user can not pass along capability access for which the user is not granted.

Exercise: Defining Administrative Capability for Users

In this exercise you make available the FTP root login feature available as a capability. This involves creating a new object in codb and defining the object's properties.

Tasks

Complete the following steps to create a administrative capability object:

1. In telnet, access the cceclient, create a CapabilityGroup object for the FTP root login feature, set the object's property values (replace the object number '60' below with the object number produce by your server appliance) and exit:

```
# /usr/sausalito/bin/cceclient
100 CSCP/0.80
200 READY
create CapabilityGroup name="adminFtpRootLogin"
104 OBJECT 60
201 OK
set 60 nameTagHelp="[[jeff-ftp.nameTagHelp]]"
201 OK
set 60 nameTag="[[jeff-ftp.nameTag]]"
201 OK
set 60 capabilities="modifyFtpRootLogin"
201 OK
bye
202 GOODBYE
```

Note – Instead of using one create command and three set commands, you could have typed one create command and set the property values on one line.

Complete the following steps to add delegate the administrative FTP root login feature to a user and verify the capability:

1. In your browser, click on Users and Groups.
2. Click the green modify pencil to the right of the user jbravo.
3. Click jbravo's Capabilities tab.
4. Select the Manage FTP Server Root Login capability and Save.

5. Log out as the `admin` user, by clicking the open door icon located in the upper right-hand side of the Server Desktop and log in as `jbravo` by entering **`jbravo`** for the username, **`suncobalt`** for the password and click Login.
6. Select the Files Services menu item.
7. In telnet, verify the `/etc/proftpd.conf` files `RootLogin` line:

   ```
   # tail -1 /etc/proftpd.conf
   ```

8. In your browser, enable the Enable Root User Login, by placing a check in the check box and click Save.
9. In telnet, verify the `/etc/proftpd.conf` files `RootLogin` line:

   ```
   # tail -1 /etc/proftpd.conf
   ```

Style Definition Files

Sausalito also enables software developers to customize the visual appearance of the Server Desktop by defining their own styles. To define the Server Desktop style, the UIFC uses an XML file. By default, the server appliance uses the /usr/sausalito/ui/style/trueBlue.xml file. By adding more XML style files, you can define additional Server Desktop styles.

The trueBlue.xml Style File

By default, the server appliance uses the file called trueBlue.xml for its style definition. This file contains the definitions for the visual appearance that the UIFC use; it is located in the /usr/sausalito/ui/style directory.

Sausalito is designed for extensibility, so do not be surprised if you find a server appliance that uses a style other than the standard style defined by the /usr/sausalito/ui/style/trueBlue.xml file.

The code below is part of the /usr/sausalito/ui/style/trueBlue.xml style file, which defines the style for the Server Desktop elements. This file defines the Server Desktop colors of backgrounds, frames, dividers, text; font size, weight and emphasis, and the images used for buttons, logo, tabs, tiles, bars, icons, arrows, and so on.

```
<style Resource name="[[trueBlue.trueBlue]]">
   <style id="Page">
         <property name="aLinkColor" value="#0033CC"/>
         <property name="backgroundImage"
value="/libImage/blocksTileSmall.gif"/>
         <property name="center" value="true"/>
         <property name="fontFamily" value="sans-serif"/>
         <property name="fontSize" value="12pt"/>
</style>
```

This course does not go into details concerning XML programming for the /usr/sausalito/ui/style/trueBlue.xml file.

Selecting Your Server Desktop Style

Additional styles are defined by the presence of more XML files than just the `trueBlue.xml` file in the `/usr/sausalito/ui/style` directory. If additional XML files are present, users can select which style they want to use in the Style drop-down list on the Account Settings page. From the Server Desktop, click Personal Profile Tab ‰ Account page. If only one XML file is present in the `/usr/sausalito/ui/style` directory, the Style drop-down list is unavailable.

Exercise: Creating and Applying Server Desktop Styles

In this exercise you adding a green color style to the Server Desktop. This is one step in the process of adding your own look and feel to the server appliance, and it is part of the extensible features of Sausalito.

Tasks

Complete the following steps to create and edit a new style file:

1. In your console window, navigate to the `/usr/sausalito/ui/style` directory, create your new style file, and open the file for editing:

   ```
   # cd /usr/sausalito/ui/style
   # cp trueBlue.xml testGreen.xml
   # vi testGreen.xml
   ```

2. Remove the reference to the True Blue style name, change some of the Server Desktop colors from blue to green, and exit the file:

   ```
   :%s/trueBlue/TestGreen/g
   :%s/FFFFFF/66CC00/g
   28 substitutions on 28 lines
   :wq
   ```

Complete the following steps to add the I18n text for your newly defined style:

1. Navigate to the /opt/*your_first_name*/i18n/po-files/en directory, create and open for editing a `TestGreen`.po file:

   ```
   # cd /opt/your_first_name/i18n/po-files/en
   # vi TestGreen.po
   ```

2. Enter the following message identifier (`msgid`) and string (`msgstr`), starting the file with one blank line on top, save and exit the file:

   [blank line, no text]
   ```
   msgid "TestGreen"
   msgstr "Test Green"
   ```

3. Save and exit the file:

   ```
   :wq
   ```

Complete the following steps to compile the `testGreen.po` file and restart the Server Desktop server to make the new text available for use.

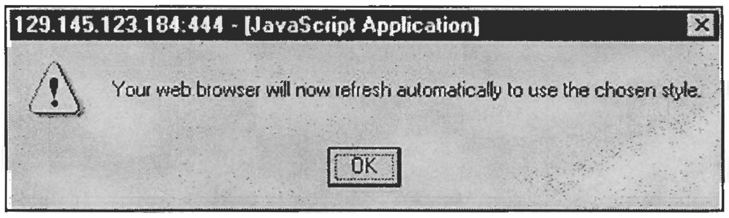
1. Compile the `.po` file into the `.mo` file and restart the admserv:

```
# msgfmt -e TestGreen.po -o
/usr/share/locale/en/LC_MESSAGES/TestGreen.mo
# /etc/rc.d/init.d/admserv restart
Stopping admin web server: ahttpd
Starting admin web server: ahttpd
#
```

Complete the following steps to enable the new style you created:

1. From your browser, log out, then log back in and navigate to the Personal Profile tab.
2. Click the Account menu item.
3. Now select the Test Green style in the Style drop-down list, and click Save.

   You should receive a pop-up window that looks like Figure 17-16:


129.145.123.184:444 - [JavaScript Application]

Your web browser will now refresh automatically to use the chosen style.

Figure 17-16 Style Change Pop-up Message

4. Click OK.

   The result should be that the text in left menu bar and in the window background on the pages is now green.

Module Wrap-up

Make sure that you can answer the following questions. See Appendix A for answers.

1. What is the key benefit that Sausalito offers for the server appliance?

Answer: ______________________________

____

2. What is the concept of object-oriented programming?

Answer: ______________________________

____

3. What are the benefits of SET transactions and GET requests?

Answer: ______________________________

____

4. What is the purpose of the Cobalt Configuration Engine (CCE)?

Answer: ______________________________

____

5. What are the responsibilities of CCE?

Answer: ______________________________

____

6. What administers Sausalito and is responsible for initiating event handlers?

Answer: ______________________________

____

7. How are objects in codb identified?

Answer: ______________________________

____

8. Is codb a database? What is its function?

Answer: ______________________________

____

9. How do services and applications register themselves in CCE?

Answer: ______________________________

____

10. How can you benefit by using event handlers?

Answer: ______________________________

____

11. What is the purpose of Type Definitions?

Answer: ______________________________

____

12. What fields are in the registration .conf files?

Answer: ______________________________

____

13. What does the handler execution stage permit?

Answer: ______________________________

____

14. What are the handler execution stages?

Answer: ____

____

15. What happens if the handler execution field is left off during the execution stage?

Answer: ____

____

16. What is the `/usr/sausalito/bin/cceclient` command used for?

Answer: ____

____

17. How do GET requests deliver HTML content to the end-user?

Answer: ____

____

18. What is the purpose of the `/usr/sausalito/ui/conf/ui.cfg` file?

Answer: ____

____

19. What kind of files are located in the `/usr/sausalito/ui/menu` directory?

Answer: ____

____

20. What tool is I18n based on in the Sausalito architecture and what benefit does it provide?

Answer: ____

____

21. How can users change their language preference?

Answer: ____

____