

## **Z!Stream Technical Overview**

A common problem faced by system administrators around the world on a daily basis: How can we rapidly, securely, and safely deploy an application to a huge user base without interfering with their computers and losing control of our own software? The potential applications of a good control system are manifold – shareware companies could allow users to try software before they buy it, IT staff could quickly unroll software across the enterprise while retaining central administrative control, collaboration teams could access programs stored on a computer on the other side of the globe. But the problem of creating a good control system that does not exact severe penalties in speed and flexibility has gone unsolved until now.

Various solutions have been tried. Most have been built around the idea of running software on a remote server while giving end users an interface to manipulate the controls, as if by proxy, and receiving images of the remote results in return. Such approaches have merit for some applications but are bandwidth-intensive, limited in functionality and not particularly responsive. They also can be severely affected by minor fluctuations in network traffic and load. If the network connection should be cut off, a program is instantly unusable. These problems have hampered serious deployment of such systems.

Z!Stream™ takes a different approach by streaming the application code itself to a client computer, which then executes it natively. This means the end user's computer runs the actual program, just as if they had bought and installed the program. But instead of being installed normally, the program resides in a special Z!Stream Virtual File System (ZVFS) disk. The program code and associated settings or initialization files are kept segregated in a proprietary Z!Stream cache file. But when in use, the program accesses and manipulates data files or documents that may be stored on the user's own hard drive, or on a network drive in a third location.

Because it executes on the local machine, a Z!Stream application is fully operational and highly responsive. It is also well insulated from network conditions. In effect, Z!Stream

allows network administrators to have their cake and eat it too, by combining the advantages of a centralized remote server with the functionality and flexibility of a local installation.

The following pages describe how Z!Stream accomplishes this complex feat, giving users the look and feel of a local installation without losing central control of the software.

### **Z!Stream Architecture**

Z!Stream is composed of several programs that work together to produce a cohesive solution. At the core of the Z!Stream architecture are Zpacks, SoftOnNet's innovative application encapsulation technology. A Zpack is constructed so that only a small part is needed at a time as it executes the contained application. These Zpacks are provided by a Container Server, which feeds a LaunchPad program on the end user's computer. The LaunchPad in turn unpacks the contents into the local Z!Stream Virtual File System, where the program then is run. Along the way, the Main Control Server, WebDeskTop and Launch Manager interact to facilitate the process.

Essentially, the Z!Stream solution is made up of two main types of components: the components of the application execution system and the management functions. These are described below.

#### **Service components**

The **Z!Stream Container Server** is the true server of the Z!Stream solution. It handles the server-side end of the application stream and communicates with the LaunchPad in real time to deliver application code as necessary. It also logs service usage and saves relevant user data and settings, and maintains a continuous dialogue with the Z!Stream database server, exchanging user and Zpack information as well as log data.

The **Z!Stream Launch Manager** handles LaunchPad version control, installation and upgrade of LaunchPad, and parameter-passing for each service application. It is the Launch Manager's job to bootstrap LaunchPad and make sure it is configured properly for the desired application.

**Z!Stream LaunchPad** is the Z!Stream local client, receiving the application stream and executing it. LaunchPad also initializes and manages the ZVFS disk and Z!Stream Virtual Registry System (ZVRS) as well as the stream cache. When a user is finished, LaunchPad saves the service application's cache data for reuse.

**Z!Stream WebDeskTop** is the Z!Stream front-end, coded in HTML and hence highly customizable. The WebDeskTop provides graphical representations of the available applications for the user to click on, providing a very user-friendly interface to Launch Manager.

### **Administrative Components**

The Z!Stream **Main Control Server** handles administrative duties for the entire Z!Stream service. These include managing users and user data, Zpack distribution and web components. The Main Control Server also provides a level of load balancing by redirecting application requests to open Container Servers.

If the administrator set usage policy at Main Control Server, each Container Server judges the policy and decide whether service is provided to user or not. Also, the Main Control Server applies needed registration, deletion or other information adjustments to Zpacks and propagates these to the affected Container Servers.

System administrators use the **Administration Tool** to manage settings across the Z!Stream service. While the Main Control Server primarily handles the job of adjusting and managing the Container Servers and other aspects of the Z!Stream solution, the Administration Tool provides a coherent, central interface to this functionality.

The **Zpack AutoEncoder** builds a Zpack file from an application, preparing it for streaming and execution by the LaunchPad. Conversely, the Z!Stream **Zpack Utility** provides needed functionality to update or alter the internal data of existing Zpacks.

## **Process Description**

When the servers are booted or rebooted and the system initializes, the Main Control Server and the Container Servers run through an initialization process before reaching a state of readiness. These processes, and the process when users execute streamed applications, are detailed below to show the interrelationships between components.

### **Main Control Server Initialization**

When Main Control Server initializes it checks a variety of settings on the Z!Stream service, including Zpack List, Container Server List, database port search, and Z!Stream servers using port search.

#### *License information reflection stage*

Licenses managing items such as expiration date, IP, and the number of multiple users are searched and compared with information found during the Control Server's initialization check. If requested services exceed the server's established limit, service is not started.

#### *Configuration information reflection stage*

Property setting items on the Main Control Server include data directory, log level settings, database usage, variety of network port setting, Container Server IP list, and expiration dates. These data are referred to when the service is in operation.

#### *Connect to DB Server stage*

If all service requirements are met, the connection takes place to the database and referencing of required data is initiated. The system refers to data on the Zpack list and to the Container Server's IP address.

#### *Zpack check stage*

During this stage a search is made of all Zpacks in the Main Control Server and the results are compared with the Zpack list found during the DB connection stage. Information on newly registered, modified or deleted Zpacks is updated at this point.

#### *Z!Stream components network port searching stage*

While the Main Control Server is in operation, it must communicate with each component. Once Zpack data has been updated, the network communication port is searched and checked for availability.

#### *Container Server connection searching stage*

An attempt is then made to access the last known connection to the Container Server, a procedure repeated every 10 seconds until a successful connection is made. Status checks are then made every three seconds, with status reports made in the appropriate field of the database for use in load balancing.

### **Container Server Initialization**

The Container Server comes online through the initialization process and the Z!Stream service is then executed through communication with the Main Control Server.

#### *Container Server Start stage*

This step starts the Container Server execution stage, first examining configuration settings. Operation is similar to steps for the Main Control Server. Additionally, a log recording process is started at this stage of operation.

#### *Zpack Check stage*

This stage initializes the Zpack repository portion of the Container Server to a ready state and verifies installed Zpack information.

#### *Connection to Database Server stage*

After verification of the information, a connection is made to the database and data is referenced from the Zpack list, the Container Server's IP address, the access log data, and user information.

#### *Z!Stream components network port searching stage*

When the Container Server operation starts, communication with each component is made, searching for necessary network communications ports and verifying availability.

## **Running a service application**

With initialization complete, an end user can run an application using the Z!Stream service.

The process of control flow interactions is explained below.

1. The end user selects an application to use through Z!Stream WebDeskTop.
2. Z!Stream WebDeskTop searches for appropriate data for the service application, from the database.
3. Z!Stream WebDeskTop transfers searched data to Launch Manager for usage.
4. Launch Manager translates and sends parameters to LaunchPad, which gathers necessary data for Z!Stream service usage.
5. LaunchPad then connects to Container server with translated data and requests application data required for service.
6. LaunchPad initializes the Z!Stream Virtual File System Disk (ZVFSD), needed for requesting required data only from Container Server, and starts Z!Stream Virtual Registry System Disc (ZVRSD).
7. LaunchPad executes service application from ZVFSD.
8. Data required for execution is processed from ZVFS. If cache data exists it is processed on the local system, otherwise it is requested from Container Server.
9. Once the application is completed, the ZVFSD and ZVRSD are each saved as Zcache Data and registry data for use in the next execution.
10. Finally, the Container Server calculates an end user's service application usage information and saves it to the appropriate database along with any billing information.

## **Implementing Z!Stream**

The Z!Stream service integrates easily with existing Web services. An existing Web server can serve up the WebDeskTop HTML and the Launch Manager ActiveX components, or this can be handled by a dedicated Z!Stream Web server. Likewise, the database server could be used to serve other company database information; for scalability and security reasons, though, it is best to keep it separate. SOFTonNET provides helpful tools to easily set up and configure these servers, either using Microsoft Windows 2000, IIS and SQL Server, or using Red Hat Linux, Apache, and MySQL.

Additionally, a server to act as the Main Control Server will be necessary, plus at least one Component Server. Again, setup is straightforward and relatively simple.

The hardware and software requirements for both Z!Stream servers and for end-user computers are shown in Table 1.

### **[TABLE 1: Z!Stream System Requirements]**

#### **Server-Side**

##### Hardware requirements

- Intel Pentium III 700Mhz or higher
- Main memory 512MB or higher (1GB recommended)
- Hard disk 10GB or higher (30GB or higher recommended)
- Network connection of T1 or higher

##### Software requirements

- Windows 2000 Server or Red Hat Linux Server OS installed
- Web server installation (IIS 5.0 or Apache™ 1.3.x)

Z!Stream requires a web server. Under Linux, the Apache web server is suitable and connects perfectly with database (MySQL) and web script language (PHP) described below. In Windows 2000 Server, the provided IIS 4.0 web server can be used; it can also be upgraded to IIS 5.0 through the Microsoft WindowsUpdate system.

- Database installation (MySQL™ 3.23.x)

Database software is necessary for effective usage of data and faster service support. MySQL is a basic Database Management System (DBMS) that SOFTonNET recommends for use with the Z!Stream service.

- Web script language installation (PHP 4.0.1)

With the basic web server supporting HTML code expected effectiveness of the Web service is not reached. To solve this problem PHP is used as Web language. The advantages of PHP is that it is usable in many OS compare to Windows supported ASP(Active Server page), effectively interlock with MySQL®, does not allow user to view source codes though web browser as server side script language, and has fast speed.

- Java™ Runtime Environment 1.3.1 installation

Z!Stream software uses Java technology internally, requiring installation of a Java Runtime Environment. In a Windows environment, version 1.3.0 is acceptable, but in Linux, version 1.3.1 or higher must be installed.

## **Client-side**

### Hardware requirements

- Intel Pentium II 300Mhz or higher (450Mhz or higher recommended)
- Main memory 64 MB or higher (128 MB or higher recommended)
- 500 MB free hard disk space or more, depending on desired service application
- Network connection 128 Mbps (downstream) or higher (xDSL, Cable Modem, etc.)

### Software requirements

- Microsoft Windows 95 or higher Windows OS
- Internet Explorer 5.0 web browser or higher (5.5 or higher recommended)
- Other software needed by the requested application (eg. DirectX 7.0 or higher, Microsoft Windows Media Player 7 or higher); optional

**[END TABLE 1]**

## **Setting up the Client System**

When a client first accesses the Z!Stream system, Z!Stream components are installed in the user's PC.

### *Z!Stream service page connection stage*

When the user connects to the service's web page, the WebDeskTop initiates the installation of Launch Manager. Since Launch Manager is an ActiveX component, the user is prompted to accept it. Launch Manager then handles version control of LaunchPad.

### *LaunchPad pre-installation and version check stage*

After being installed, Launch Manager checks to see if LaunchPad needs to be installed. If there is no installed version or the version is not up to date, Launch Manager will install the correct version of LaunchPad.

### *LaunchPad installation stage*

If LaunchPad has to be installed, Launch Manager downloads the LaunchPad installation file from the appropriate site and the InstallShield wizard starts up to guide the user through the installation process. After installation, the user's selected service application is executed.

## **Creating and Modifying Zpacks**

The Zpack AutoEncoder simplifies the process of creating a Zpack from an installed program. The system administrator can then use the Zpack Utility to modify Zpacks, to apply patches, change settings, or update registration and license information. All such changes are made centrally, which tremendously simplifies management.

## **Deploying Zpacks**

Deploying Zpacks for use by the Z!Stream service is a two-stage process: First, the Zpack is registered in the Main Control Server; second, the Main Control Server installs the Zpack on relevant Container Servers.

### *Zpack transmission stage using Administration Tool*

The system administrator uses the Administration Tool to transmit a new Zpack to the Control Server, using the RMI protocol. The Main Control Server saves the Zpack in data directory and updates Zpack list of management purpose data. Since the process uses Java technologies, registration is quick and easy.

### *Zpack data analysis, saving in DB stage*

The Main Control Server analyzes the Zpack and saves necessary Zpack data in the database using XML. At this time the Zpack receives a unique Zpack ID.

### *Command of registration in Container Server stage*

The Main Control Server now transmits the Zpack to the Container Server, using its internal MFPT protocol. The Container Server installs the Zpack in the appropriate data directory. The Main Control Server then updates the Zpack list in the database. The system is now ready for use.

## **Securing Z!Stream**

Obviously, such a powerful, flexible system presents some potential for abuse. However, Z!Stream is equipped with safeguards to prevent software theft or piracy and to protect the system from attack and exploitation by malicious hackers.

### *License control*

One main function of the Main Control Server is centralized license control. The Main Control Server can veto service if it determines the user is not authorized to use an application on the basis of their specific IP address, or if there are not enough valid unexpired product licenses to instantiate another copy of the program.

#### *Code integrity*

The executable codes that are transmitted from Z!Stream server to client computers are in an encoded form that prevents a hacker from disassembling or reverse engineering code from Z!Stream cache. The code (cached or streamed) operates only under a virtual file system of Z!Stream.

#### *Transmission security*

Unlike terminal servers where the client computers transmits data to server, Z!Stream clients only sends code transmission requests to the server and receives only the executable codes from the server. Therefore, user data is not exposed from client to server.

#### *Authentication*

As discussed above, the Main Control Server verifies each client's eligibility for service before authorizing a stream. The user authentication between the client and Z!Stream server is done through SSL encryption.

### **Performance**

The only Z!Stream components to actually affect perceived performance are the Container Server and LaunchPad. Due to its distributed nature, it is difficult to predict the application-specific performance of the Z!Stream system, but it is reasonable to conclude that the maximum number of simultaneously open streams is a good measure of overall system performance.

This figure will vary, depending upon the hardware and software specifications of the server and, specifically, the amount of installed RAM, the power of the server's CPU, and the server's underlying operating system. However, the results of tests with public confidence show that the number of simultaneous connection possible sessions is about 1000 per server – literally orders of magnitude above competitive systems.

If several Container Servers are in use, the Z!Stream system is capable of handling load balancing between them without the need for a hardware-based load balancing function such as a Layer 4 Switch. Therefore there is no real limit to the maximum number of users that a Z!Stream service can handle.

Z!Stream's best-of-breed design makes it the superior choice, combining the robust look and feel of local installation with the economical and security benefits of remote execution. Table 2 clearly shows the advantages of Z!Stream over local installation methods; Table 3 details the benefits of the Z!Stream solution as opposed to the remote execution model of our competitors.

**[TABLE 2: Z!Stream vs. Local Installation]**

- *Time to deploy:* Local installation requires transmitting the entire application program to the end user's computer, and full application installation, before any functionality can be accessed. Z!Stream allows the user to have full access to any component within moments.
- *Stability:* To determine the presence or nature of any hardware or software conflicts, often one must actually install the software and attempt to use it. With local installation, this can cause conflicts, problems, or system instabilities to develop, which may be difficult to resolve even after the program is uninstalled. Z!Stream helps by isolating the program in a virtual drive, with a virtual registry, so that any conflicts or incompatibilities will not disturb the local system.
- *Cost of maintenance:* Since all the Zpacks are stored and administered centrally, it is easy for a Z!Stream administrator to adjust settings or apply updates across the entire user base. Conversely, with a local installation, the administrator must tailor updates or patches to the individual setups of different user groups, and then travel to the physical location of each computer or else rely on third-party remote administration software.
- *Security:* When software is locally installed on a user's computer, it is difficult to control their use of it. A sophisticated hacker could steal the program or perhaps reverse engineer and modify it, for instance circumventing certain restrictions or security measures. This can be a serious concern for distributors of shareware and other software. By contrast, Z!Stream secures the executable code and

makes it much more difficult – if not impossible – for malicious users to bypass or disable security restrictions.

**[END TABLE 2]**

**[TABLE 3: Z!Stream vs. Remote Execution]**

- *Functionality:* When software is executed remotely, on a server, certain functionality that requires interaction with components of a user's computer is greatly hindered or even disabled. This can even make it difficult for a user to access their local files with the software – a potentially fatal shortcoming. Other functionality too may be affected. With Z!Stream, since the program actually runs on the user's computer, it is fully functional and can access local any resources just as if the user had installed it locally.
- *Responsiveness:* Z!Stream applications execute natively on the user's computer, with very little performance overhead. The user experience is thus responsive and impressive. By contrast, remotely executed programs' responsiveness may vary with network conditions and provide an unsatisfactory user experience.
- *Server load:* Depending on the type and use of an application, software can place a significant processor load on the computer running it. When a server is executing programs for a variety of clients, this can lead to reduced responsiveness and poor performance – not to mention the increased amount of server hardware and software needed to sustain this heavier load. Since Z!Stream servers merely transfer data to and from clients, however, their load is analogous to that of an ordinary Web or File server. With a large user base or processor-intensive applications, the Z!Stream solution could save a company many thousands of dollars in additional server costs.

**[END TABLE 3]**

###