CSP 1.4: An HTTP-Based Protocol for Parameterized, Aggregated Content

This document was modified: 6/25/2007

1. Introduction

1.1. Motivation and Design Goals

Content servers are an important part of the distributed AquaLogic User Interaction (ALUI) architecture. They serve up content to gateway servers to be included in personalized pages and web services. This document addresses the communication protocol to be used between a gateway server and a content server. CSP has the following design goals:

- Platform independence (a content server should be able to run on any major web server)
- Standards compliance
- Ease of implementation for content servers
- Facilitation of a robust caching architecture
- Statelessness on the part of the content server
- Transparency of content use (i.e., a web server acting as a content server should be able to use the same protocol and logic for all kinds of content, including pagelets, preference pages, images, and associated pages)

CSP is an extension to HTTP 1.1. The gateway server acts as a gateway for data served up by a content server, as defined in RFC 2616 (http://www.ietf.org/rfc/rfc2616.txt). This means that when a client asks a gateway server for a URI, the client will retrieve the requested resource from the content server. The client will NOT know of the existence of the content server; as far as the client is concerned, the proxy server is the origin server for the resource.

The main hurdles that this protocol attempts to address are:

- How to send preference information to content servers in a robust, standards-compliant, platform independent, and consistent manner.
- How to send information about the gateway environment in a standards-compliant manner that is easy for a developer to use.
- How to enable caching of pagelet content on both the client browser and the gateway in a standards-compliant manner to facilitate scalability.
- How to deal with complications involving caching and header meta-information arising from concatenating pieces of content from multiple sources into a single piece of content.

Note: Although this specification is written as an extension to HTTP 1.1 (as described in RFC 2616), there is no restriction that a content server must run HTTP 1.1 to take advantage of this protocol.

1.2. Glossary of Terms

For generic terms not listed here and an explanation of notational conventions used in this document, see RFC 2616 (sections 1.3 and 2.1).

CSP: As of version 1.2, the official name of the protocol was changed from “Content Server Protocol” to “CSP.”

Content Server: A server that serves up (over HTTP) parameterized content that is designed to be aggregated. As far as this specification is concerned, the content server is the origin server for the content, but this does not preclude a configuration in which the content server is a gateway to one or more other origin servers. A content server is a Producer as defined by WSRP. A content server is represented by a remote server object in AquaLogic Interaction, and by an application resource object in Ensemble.

Gateway Server: A gateway, as defined by RFC 2616 (http://www.ietf.org/rfc/rfc2616.txt), that acts as an HTTP client of one or more content servers. A gateway server stores and retrieves parameters for pagelets as preference information. Gateway servers often aggregate HTTP responses from content servers into a single HTTP response to the client. The AquaLogic Interaction portal server gateway and the Ensemble proxy are both gateway servers.

Pagelet: A particular resource represented by an object on the gateway server and associated content rendered by the content server. In AquaLogic Interaction, a pagelet is usually represented by a **portlet** object, but may be another type of object such as a document or a federated search. In previous versions of this protocol, pagelets were called **gadgets**; some header and variable names use this term for backwards-compatibility.

Realm: Any logical grouping of one or more resources on the gateway server. Realms are used for sharing preference information between different users that are aggregated into the same resource. Realms in this specification are similar to the concept of realms in HTTP Authentication, as defined in RFC 2617 (http://www.ietf.org/rfc/rfc2617.txt). The difference is that they are defined by the client (gateway server) and are opaque to the origin server (content server) rather than vice versa. The semantics of what a realm means may be different depending on the implementation of the gateway server; in AquaLogic Interaction, realms are implemented as “Communities.”

2. Protocol Overview

CSP must communicate eight basic sets of information to the content server to make it possible for the server to effectively generate content. There are six types of pagelet-specific preferences, as well as gateway configuration variables and user information. (As noted above, gadget is the legacy term for pagelet.)

- **Global-Gadget level preferences**
  Global-Gadget level preferences apply to a pagelet wherever the gateway server uses it, no matter how it is accessed. These preferences are settings known to the administrator, but transparent to the user. For example, in the case of a pagelet that queries a database, the database location and login would be Global-Gadget level preferences. These variables are modifiable by the content server, only if the current user has the appropriate security permissions.

- **Gadget-Realm-User level preferences**
  Gadget-Realm-User level preferences allow personalization of a pagelet for the user who is viewing it. Unlike User level or Realm level preferences, this information will never be sent to any other pagelet on this or any other content server. These variables are modifiable by the content server.

- **User level preferences**
  User level preferences let multiple pagelets across all realms share personalization information for the user who is viewing them. This type of preference may be shared with other pagelets on this or other content servers. These variables are modifiable by the content server.
  User level preferences are useful when a pagelet developer is creating a suite of pagelets that will need to share personalized information. For example, a mail and calendar applications may share login information to a groupware program such as Exchange or Notes. When a user changes the login preferences for one pagelet in the suite, it changes automatically for any others that share that User level preference.

- **Realm level preferences**
  Realm level preferences depend on how the gateway server aggregates content. The gateway server may display the pagelet as part of several different resources. This specification allows a gateway server to define a group of those resources as a realm, which will share a set of preferences. These variables are modifiable by the content server, only if the current user has the appropriate security permissions.

  Realms are similar to the concept of realms in the HTTP Authentication specification (RFC 2617). The semantics and scope of a realm is up to the gateway server and is opaque to the content server. A gateway server MAY define every resource it serves up to be a separate realm (in which case realm level preferences are functionally just resource or page level preferences). Conversely, a gateway server MAY also define every resource it serves up to be in the same realm (in which case realm level preferences are site level preferences). ALL pagelets and ALL users on a gateway server have access to Realm level preferences.

- **Gadget-Realm level preferences**
  Gadget-Realm level preferences apply to the pagelet within the realm in which it is aggregated. These preferences are shared by all users who access the pagelet in the same realm. These variables are modifiable by the content server, only if the current user has the appropriate security permissions.

- **Session level preferences**
  Session level preferences are just like user preferences, but are stored on the session and not persisted in the portal database. These preferences are scoped per user, and sent to pagelets exactly the same way as user preferences. Session preferences can be set by pagelets through the IDK or via JavaScript through the AquaLogic Interaction Scripting Framework. If a user logs out or his session expires, he loses all session preferences.

- **Gateway configuration variables**
  Gateway configuration variables let the content server know the kind of environment in which content will be displayed. These arguments include things like URIs of pages to redirect to when done, logged-in user names, and identifiers of objects in the gateway. These variables are not modifiable by the content server.

- **User Information**
  User Information allows administrators to pass properties about a user, such as e-mail address, employee ID, or department, from centralized repositories like LDAP, SSO, or company databases to pagelets. These variables are not modifiable by the content server.

These eight sets of information are sent in the HTTP Request headers from the gateway server to the content server. If and when the content server wishes to change these variables, it sends information back to the gateway server in the HTTP Response headers. This extension to standard HTTP helps achieve the goal of transparency in the protocol.

All extensions to HTTP are based on the extensibility of HTTP headers as defined in RFC 2616. The main addition is to the grammar rule that defines the extension-header element (section 7.1). CSP modifies that rule to read:

```
extension-header  =  csp-header | message-header

csp-header        =  csp-prefs | csp-gateway-config
```

Certain CSP headers must be encoded to allow the transmission of Unicode character values in HTTP headers. The encoding rules are defined as follows:

```
csp-encoded-value =     0*<ALPHA  | DIGIT | “:” | “/” | “?” |
                        “.”  | “\”   | “_” | “-“ | csp-percent-u>

csp-percent-u = (“%” “u” HEX HEX HEX HEX)
```

3. Preferences: Sending and Storing Parameters for the Content Server

In order to serve up dynamic, useful content, a content server must be able to accept parameters from the gateway server and store those parameters on the gateway server for future use. These parameters are referred to as "preferences." AquaLogic Interaction has added HTTP headers to communicate this information between the content server and gateway server.

The new HTTP headers are extra HTTP headers defined in this specification:

```
csp-prefs   =     csp-set-prefs | csp-get-prefs | csp-can-set


csp-get-prefs     =     csp-get-global-prefs |
                        csp-get-gadget-realm-prefs |
                        csp-get-gadget-realm-user-prefs |
                        csp-get-user-prefs | csp-get-realm-prefs
                        csp-get-session-prefs | csp-get-user-info

csp-set-prefs     =     csp-set-global-prefs |
                        csp-set-gadget-realm-prefs |
                        csp-set-gadget-realm-user-prefs |
                        csp-set-user-prefs | csp-set-realm-prefs
                        csp-set-session-prefs

csp-can-set       =     csp-can-set-none | csp-pref-level
```

These headers present a comma delimited list of one or more name-value pairs (specific header names are detailed in section 3.3).

3.1. Sending Preferences to the Content Server

The gateway server MUST send all preferences to the content server in a Request header:

```
request-header    =     csp-get-prefs ":" 1#csp-name-value-pair

csp-get-prefs     =     csp-get-global-prefs |
                        csp-get-gadget-realm-prefs |
                        csp-get-gadget-realm-user-prefs |
                        csp-get-user-prefs | csp-get-realm-prefs
                        csp-get-session-prefs | csp-get-user-info

csp-name-value-pair =   csp-name "=" csp-value
                        *(";" csp-pref-metadata)

csp-name          =     token

csp-value         =     word

csp-pref-metadata =     csp-name "=" csp-value
```

Each name-value pair represents one particular preference. How the gateway server determines which settings are applicable to the content server is a quality of implementation issue and is not covered in this protocol.

Note: There are currently no defined metadata name-value pairs associated with preferences (i.e., “name=value; metadata1=metadatavalue1; metadata2=metadatavalue2” as in the charset parameter in the HTTP Content-Type header). In future versions of this specification, there might be a need for preference metadata, which is the purpose of the csp-pref-metadata non-terminal. Parsers should ignore any semi-colon delimited metadata in order to be forward-compatible with versions of this protocol which may use such metadata.

A content server MAY ignore preferences if it does not care about their values.

3.2. Storing Preferences on the Gateway Server

To store preferences on the gateway server, the content server MUST use the appropriate Response header:

```
response-header    =     csp-set-prefs ":" 1#csp-name-value-pair

csp-set-prefs      =     csp-set-global-prefs |
                         csp-set-gadget-realm-prefs |
                         csp-set-gadget-realm-user-prefs |
                         csp-set-user-prefs | csp-set-realm-prefs
                         csp-set-session-prefs
```

These Response headers present a comma delimited list of one or more name-value pairs which the gateway server MUST store as the appropriate type of preference. If a preference with the specified name already exists, the gateway server MUST overwrite it with the new value. The gateway server MUST interpret a name-value pair with a value of whitespace as a directive to delete the preference keyed on the name.

A content server SHOULD be very careful when storing a User or Realm preference to ensure that the name of the preference will be unique. The fact that these preferences are shared among pagelets means that there is a potential for name collisions, and those name collisions could have unexpected consequences.

3.3. Preference Types

There are six types of pagelet-specific preferences, as described in section 2. User Information is accessed in the same manner as preferences, and is also covered below (3.3.7).

3.3.1. Global-Gadget Level Preferences

Global-Gadget level preferences (“Administrative preferences”) are sent to the content server via the CSP-Global-Gadget-Pref header:

```
csp-get-global-prefs  =  "CSP-Global-Gadget-Pref" ":"
                         1#csp-name-value-pair
```

Global-Gadget level preferences are set on the gateway server through the CSP-Set-Global-Gadget-Pref header.

```
csp-set-global-prefs  =  "CSP-Set-Global-Gadget-Pref" ":"
                         1#csp-name-value-pair
```

3.3.2. Gadget-Realm-User Level Preferences

Gadget-Realm-User level preferences ("Portlet preferences") are sent to the content server via the CSP-Gadget-Realm-User-Pref header:

```
      csp-get-gadget-realm-user-prefs  = "CSP-Gadget-Realm-User-Pref"
                        ":" 1#csp-name-value-pair
```

Gadget-Realm-User level preferences are set on the gateway server through the CSP-Set-Gadget-Realm-User-Pref header:

```
    csp-set-gadget-realm-user-prefs = "CSP-Set-Gadget-Realm-User-Pref"
                        ":" 1#csp-name-value-pair
```

3.3.3. User Level Preferences

User level preferences ("User preferences") are sent to the content server via the CSP-User-Pref header:

```
      csp-get-user-prefs  =  "CSP-User-Pref" ":" 1#csp-name-value-pair
```

User level preferences are set on the gateway server through the CSP-Set-User-Pref header:

```
      csp-set-user-prefs  =  "CSP-Set-User-Pref" ":"
                              1#csp-name-value-pair
```

3.3.4. Gadget-Realm Level Preferences

Gadget-Realm level preferences ("CommunityPortlet preferences") are sent to the content server via the CSP-Gadget-Realm-Pref header:

```
      csp-get-gadget-realm-prefs  =  "CSP-Gadget-Realm-Pref" ":"
                              1#csp-name-value-pair
```

Gadget-Realm level preferences are set on the gateway server through the CSP-Set-Gadget-Realm-Pref header.

```
      csp-set-gadget-realm-prefs  =  "CSP-Set-Gadget-Realm-Pref"
                        ":" 1#csp-name-value-pair
```

3.3.5. Realm Level Preferences

Realm level preferences ("Community preferences") are sent to the content server via the CSP-Realm-Pref header:

```
      csp-get-realm-prefs  =  "CSP-Realm-Pref" ":"
                              1#csp-name-value-pair
```

Realm level preferences are set on the gateway server through the CSP-Set-Realm-Pref header.

```
      csp-set-realm-prefs  =  "CSP-Set-Realm-Pref" ":"
                              1#csp-name-value-pair
```

3.3.6. Session Level Preferences

Session level preferences (“Session preferences”) are sent to the content server via the CSP-Session-Pref header:

```
csp-get-session-prefs  =  "CSP-Session-Pref" ":"
                         1#csp-name-value-pair
```

Session level preferences are set on the gateway server through the CSP-Set-Session-Pref header.

```
csp-set-session-prefs  =  "CSP-Set-Session-Pref" ":"
                         1#csp-name-value-pair
```

3.3.7. User Information

User Information is sent to the content server via the CSP-User-Info header:

```
csp-get-user-info =  "CSP-User-Info" ":" 1#csp-name-value-pair
```

User Information cannot be modified by the content server.

3.4. Preference Permissions

Shared preferences create complexity in the permissions that determine which users are allowed to modify specific preferences. The gateway server SHOULD send a CSP-Can-Set header to a content server that tells the content server which levels of preferences it has permission to modify.

```
csp-can-set      =      "CSP-Can-Set" ":" csp-can-set-none |
                        1#csp-pref-level

csp-can-set-none =      "None"

csp-pref-level   =      "Realm" | "User" | "Gadget-Realm" |
                        "Global-Gadget" | "Gadget-User"
```

The CSP-Can-Set request header is a comma delimited list of descriptors that tells the content server which levels of preference it has permission to store.

- If the list does not contain "Realm" the content server SHOULD NOT use the CSP-Set-Realm-Pref header.
- If the list does not contain "User" the content server SHOULD NOT use the CSP-Set-User-Pref header.
- If the list does not contain "Global-Gadget" the content server SHOULD NOT use the CSP-Set-Global-Gadget-Pref header.
- If the list does not contain "Gadget-Realm" the content server SHOULD NOT use the CSP-Set-Gadget-Realm-Pref header.
- If the list does not contain "Gadget-User" the content server SHOULD NOT use the CSP-Set-Gadget-Realm-User-Pref header.
- If there is no CSP-Can-Set header in the request, the content server MUST treat the request AS IF a header of "CSP-Can-Set: User, Gadget-User" was sent.

If a content server tries to store a preference that it does not have permission to set, the behavior of the gateway server is undefined by this specification. Because there is not another request made to the content server, the gateway server cannot easily transmit error information. The gateway server SHOULD silently fail in the case of storage failures, although a gateway server MAY choose to store the settings as another form of preference. (Session preferences can always be set.)

4. Gateway Configuration Variables

To communicate to the content server what type of content it should return, the gateway server sends a specific set of configuration variables describing the environment in which the content will be used.

There is a group of configuration variables that MUST be sent by the gateway server, and an extensible set of variables that MAY be sent by the gateway server. Any content server MAY choose to ignore any of the gateway configuration variables, although this may have unexpected results if the gateway server is using the content in a way that the content server does not expect.

The grammar rule that defines the gateway configuration headers is:

```
csp-gateway-config      =     csp-protocol-version    |
                              csp-gateway-type        |
                              csp-session-token       |
                              csp-activity-rights     |
                              csp-gateway-specific-config
```

4.1. CSP-Protocol-Version

The CSP-Protocol-Version header describes the protocol version to which the gateway server conforms.

```
csp-protocol-version    = "CSP-Protocol-Version" ":" csp-version-number

csp-version-number      = csp-version-major "." csp-version-minor

csp-version-major       = 1*DIGIT

csp-version-minor       = 1*DIGIT
```

For this version of the protocol, the major version MUST be "1", and the minor version MUST be "4". Inclusion of this versioning parameter guarantees neither that there will be any other versions of this protocol produced nor that any particular versioning scheme will be used for the protocol (i.e., the next version of the protocol could be "1.5", "2.0", or even "12.345").

4.2. CSP-Aggregation-Mode

The CSP-Aggregation-Mode header describes what sort of aggregation the gateway plans to apply to the content from the content server.

```
csp-aggregation-mode = "CSP-Aggregation-Mode" ":"
csp-aggregation-mode-name
csp-aggregation-mode-name = "Multiple" | "Hosted" | "Single"
```

See section 6.1 for a definition of aggregation types. The gateway server MUST send the CSP-Aggregation-Mode header.

4.3. CSP-Gateway-Type

The CSP-Gateway-Type header describes what type of gateway the gateway server is, which tells the content server what set of gateway specific headers it will send.

```
csp-gateway-type  =  "CSP-Gateway-Type" ":" csp-gateway-type-name

csp-gateway-type-name  =  token
```

The following gateway types are defined:

CSP-Gateway-Type	Gateway Server
Plumtree	AquaLogic Interaction
Proxy	Ensemble

The gateway server MUST send the CSP-Gateway-Type header, but whether or not the gateway server needs to send any other gateway-specific headers depends on the semantics of the gateway type. For the Plumtree gateway, there is a group of mandatory headers (described in section 4.6 CSP-Gateway-Specific-Config, below).

4.4. CSP-Session-Token

The CSP-Session-Token header contains a secure token that can be used to connect a new session on the content server as the current user on the gateway server sending the request.

```
   csp-session-token  =  "CSP-Session-Token" ":" quoted-string
```

The gateway server MAY send the CSP-Session-Token header or omit it, and care should be taken to ensure that the header is sent only over secured connections to trusted content servers.

4.5. CSP-Activity-Rights

The CSP-Activity-Rights header enumerates a list of rights possessed by the current user.

```
   csp-activity-rights  =  "CSP-Activity-Rights " ":" 1#csp-encoded-value
```

The gateway server MAY send the CSP-Activity-Rights header or omit it.

4.6. CSP-Gateway-Specific-Config

The CSP-Gateway-Specific-Config header is designed to allow the gateway to send application-specific information to the content server.

The configuration information is sent from the gateway server to the content server as a comma-delimited list of name-value pairs. The semantics of these names and their values is negotiable between the gateway server and the content server. This mechanism allows content servers to customize content for specific implementations of this specification.

```
   csp-gateway-specific-config   = "CSP-Gateway-Specific-Config" ":"
                                       1#csp-gateway-config-pair

   csp-gateway-config-pair       =  csp-gateway-config-name "="
                                       csp-gateway-config-value |
                                       pt-config-pair

   csp-gateway-config-name       =  token

   csp-gateway-config-value      =  word
```

A Plumtree gateway (specified in the CSP-Gateway-Type header) MUST send a group of AquaLogic Interaction-specific arguments to the content server to help it create content that will integrate well into a portal page.

```
   pt-config-pair =  pt-user-name       |
                     pt-user-id         |
                     pt-stylesheet-uri  |
                     pt-hostpage-uri    |
                     pt-community-id    |
                     pt-gadget-id       |
                     pt-gateway-version |
                     pt-content-mode    |
                     pt-return-uri      |
                     pt-time-zone       |
                     pt-imageserver-uri |
                     pt-user-charset    |
                     pt-page-id         |
                     pt-community-acl   |
                     pt-soap-api-uri    |
                     pt-portal-uuid     |
                     pt-class-id        |
                     pt-ui-type         |
                     pt-subportal-id    |
                     pt-alignment       |
                     pt-guest-user      |
                     pt-identifier
```

4.6.1. PT-User-Name

PT-User-Name is the user name of the currently logged in user.

```
      pt-user-name      =  "PT-User-Name" "=" csp-encoded-value
```

4.6.2. PT-User-ID

PT-User-ID is the ID of the currently logged in user.

```
      pt-user-id        =  "PT-User-ID" "="     pt-user-id-value

      pt-user-id-value  =  1*DIGIT
```

4.6.3. PT-Stylesheet-URI

PT-Stylesheet-URI is the absolute URI of the current stylesheet.

```
      pt-stylesheet-uri  =  "PT-Stylesheet-URI" "=" absoluteURI
```

4.6.4. PT-Hostpage-URI

PT-Hostpage-URI is the absolute URI of the page that will host the content from the content server.

```
      pt-hostpage-uri   =  "PT-Hostpage-URI" "=" csp-encoded-value
```

4.6.5. PT-Community-ID

PT-Community-ID is the ID of the Community (i.e., realm) in which the content served up by the content server will be displayed (0 if on a MyPage).

```
      pt-community-id   = "PT-Community-ID" "=" pt-community-id-value

      pt-community-id-value  =  1*DIGIT
```

4.6.6. PT-Gadget-ID

PT-Gadget-ID is the ID of the object being served up by the content server. The class of object is specified in PT-Class-ID (4.6.17).

```
pt-gadget-id  =  "PT-Gadget-ID" "=" pt-gadget-id-value

pt-gadget-id-value  =  1*DIGIT
```

4.6.7. PT-Gateway-Version

PT-Gateway-Version is the version of the gateway specification to which the arguments adhere.

```
pt-gateway-version  =  "PT-Gateway-Version" "="
                              pt-gateway-version-value

pt-gateway-version-value = pt-gateway-version-major-value \
                              "." pt-gateway-version-minor-value

pt-gateway-version-major-value  =  1*DIGIT

pt-gateway-version-minor-value  =  1*DIGIT
```

For this version of the gateway specification, the major version MUST be "1", and the minor version MUST be "0". Inclusion of this versioning parameter guarantees neither that there will be any other versions produced nor that any particular versioning scheme will be used (i.e., the next version of the protocol could be "1.1", "2.0", or even "12.356"). Note: There is no implied or explicit connection between the values of CSP-Protocol-Version and PT-Gateway-Version.

4.6.8. PT-Content-Mode

PT-Content-Mode is the mode under which the pagelet is running, from the enumeration PT_GADGET_MODES.

```
pt-content-mode  =  "PT-Content-Mode" "=" pt-content-mode-value

pt-content-mode-value  =  1*DIGIT
```

4.6.9. PT-Return-URI

PT-Return-URI is the absolute URI of the page in the portal that the pagelet should redirect users to when they have finished a task, e.g., setting personal settings. It will generally be the URI of the page that aggregates the pagelet itself.

```
pt-return-uri  =  "PT-Return-URI" "=" absoluteURI
```

4.6.10. PT-Time-Zone

PT-Time-Zone is the current time zone in the format used by the portal. In previous versions of the portal, this header was sent as a number. This format is now deprecated. The gateway server will send the current user’s time zone in the format defined by the timezone database (http://www.twinsun.com/tz/tz-link.htm). Optional: may not be sent in some requests.

```
pt-time-zone  =  "PT-Time-Zone" "=" pt-time-zone-value

pt-time-zone-value  =  csp-encoded-value or 1*DIGIT
```

4.6.11. PT-Imageserver-URI

PT-Imageserver-URI is the absolute URI of the server that hosts images for the portal.

```
pt-imageserver-uri  =  "PT-Imageserver-URI" "=" absoluteURI
```

4.6.12. PT-User-Charset

PT-User-Charset is the character encoding chosen by the user for use in the portal. HTTP character sets are identified by case-insensitive tokens. The complete set of tokens is defined by the IANA Character Set registry (http://www.ietf.org/rfc/rfc1700.txt pp. 100-116).

```
pt-user-charset  =  "PT-User-Charset" "=" charset
```

Although HTTP allows an arbitrary token to be used as a charset value, any token that has a predefined value within the IANA Character Set registry MUST represent the character set defined by that registry.

4.6.13. PT-Page-ID

PT-Page-ID is the ID of the page in which the content served up by the content server will be displayed. PT-Page-ID is optional and may not be sent in some requests.

```
pt-page-id  = "PT-Page-ID" "=" pt-page-id-value

pt-page-id-value  =  1*DIGIT
```

4.6.14. PT-Community-ACL

PT-Community-ACL is the access level of the current user to the Community (i.e., realm) in which the content served up by the content server will be displayed. PT-Community-ACL is optional and may not be sent in some requests.

```
pt-community-acl  = "PT-Community-ACL" "=" pt-community-acl-value

pt-community-acl-value  =  1*DIGIT
```

4.6.15. PT-SOAP-API-URI

PT-SOAP-API-URI is the absolute URI of the SOAP listener that the content server can use to send calls back to the gateway server API.

```
pt-soap-api-uri   =  "PT-SOAP-API-URI" "=" absoluteURI
```

4.6.16. PT-Portal-UUID

PT-Portal-UUID uniquely identifies the installation of the gateway server sending the request.

```
pt-portal-uuid    = "PT-Portal-UUID" "="  quoted-string
```

4.6.17. PT-Class-ID

PT-Class-ID is the class ID of the object being served up by the content server.

```
pt-class-id       = "PT-Class-ID" "="  pt-class-id-value

pt-class-id-value  =  1*DIGIT
```

The table below lists the AquaLogic Interaction objects and class IDs that can be accessed via the gateway. **Note:** These IDs are specific to the current version of the portal; new IDs may be added in future versions without a change to the CSP specification.

Object Type	Class ID
Data Source	35
Federated Search	46
Portlet	43
Portlet Template	61
Web Service	47

4.6.18. PT-UI-Type

PT-UI-Type identifies the type of UI in which the content served up by the content server will be displayed, for example a wireless device. Optional; may not be sent in some requests.

```
pt-ui-type  =  "PT-UI-Type" "=" quoted-string
```

4.6.19. PT-Subportal-ID

PT-Subportal-ID identifies the ID of the experience definition in which the content served up by the content server will be displayed. PT-Subportal-ID is optional and may not be sent in some requests (this header can be enabled in the Web Service editor in the portal).

```
pt-subportal-id    =  "PT-Subportal-ID" "=" pt-subportal-id-value

pt-subportal-id-value  =  1*DIGIT
```

4.6.20. PT-Alignment

PT-Alignment identifies the pagelet alignment. PT-Alignment is optional and may not be sent in some requests (this header can be enabled in the Web Service editor in the portal). Alignment is only sent in multiple aggregation mode (see section 6.1). If the aggregation mode is single, PT-Alignment is not included in the CSP-Gateway-Specific-Config header.

```
pt-alignment      =  "PT-Alignment" "="  pt-alignment-value

pt-alignment-value  =  1*DIGIT
```

The table below lists the alignments that can be assigned to a pagelet.

Alignment	ID
Narrow	1
Wide	2
Header	3
Footer	4
Content Canvas	5

4.6.21. PT-Guest-User

PT-Guest-User identifies whether the current user is a guest user in the portal. If the value is 1, the user is a guest; if the value is 0, the user is not a guest user.

```
pt-guest-user     =  "PT-Guest-User" "=" pt-guest-user-value

pt-guest-user-value  =  1*DIGIT
```

4.6.22. PT-Identifier

PT-Identifier is the name of the current object (pagelet, portlet, page, etc.).

```
pt-identifier     =  "PT-Identifier" "=" csp-encoded-value
```

5. Content Display Variables

To communicate to the gateway server how the content it returns should be displayed, there is a group of variables that MAY be sent by the content server. Any gateway server MAY choose to ignore any of the content display variables, although the content may display in a way that the content server does not expect.

The grammar rule that defines the content display headers is:

```
csp-content-display     =     csp-display-mode        |
                              csp-title-bar           |
                              ptgw-streaming
```

5.1. CSP-Display-Mode

The CSP-Display-Mode header indicates whether the content returned should display in Hosted or Single aggregation. The gateway server MUST ignore the value of this header when it displays the content in Multiple aggregation mode.

```
csp-display-mode  = "CSP-Display-Mode" ":" csp-aggregation-type

csp-aggregation-type   =  "Hosted" | "Single"
```

5.2. CSP-Title-Bar

The CSP-Title-Bar header indicates a preferred title to display above the content.

```
csp-title-bar     = "CSP-Title-Bar" ":" csp-encoded-value
```

5.3. PTGW-Streaming

The PTGW-Streaming header directs the gateway server to stream the content through to the client without any link fixing or other content modification.

```
ptgw-streaming    = "PTGW-Streaming" ":" ptgw-streaming-enabled

ptgw-streaming-enabled   =  "yes" | "no"
```

6. Gateway Considerations

An integral part of this specification is the idea that the gateway server acts as a gateway that aggregates content. There are special considerations that stem from this characteristic of the specification and that have implications for the protocol. This section explores the complications that result from the gateway server acting as a gateway and aggregating content.

6.1. Aggregation Type

The gateway server aggregates content based on the source of the content. There are three basic aggregation types:

6.1.1. Multiple

Content that is part of a portal page (MyPage or Community) will be aggregated into an HTTP response with content from other content servers, and optionally content from the gateway server. The headers for the aggregated response will be constructed by the gateway server, and the content server MUST NOT expect that its headers will pass through to the client.

6.1.2. Hosted

Content from preference pages or other gateway pages that also display content from the gateway server (e.g., a portal banner) will be aggregated into an HTTP response with content from the gateway server, but no content from other content servers will be used. An example of the Hosted aggregation type is a "preview" type page, which might show a piece of content with a standard "Close Window" button at the end. The gateway server SHOULD send through the content server's HTTP headers, subject to the restrictions of section 5.7.

6.1.3. Single

Content from preference pages or other gatewayed pages that do not include any content from any other servers will not be aggregated by the gateway server. The gateway server MAY transform the content from the content server to perform operations such as HTML link fixing, XML transforms, etc., as described in section 5.2. The gateway server MUST send through the content server's HTTP headers, subject to the restrictions of section 5.7.

6.2. Link Fixing and Other Content Modification

The content server should be aware that its content may be modified by the gateway server in its role as a gateway. This MAY include (but is not limited to) operations such as HTML link fixing or translation of format (e.g., translating XML to HTML using a stylesheet). The actual transformations provided by the gateway server to gateway the content are not defined in this specification, but the fact that content modification occurs informs how headers from the content server are passed through to the client (see sections 5.4 and 5.7).

6.3. Caching

RFC 2616 provides thorough guidelines and requirements for HTTP caching. This specification does not redefine these requirements, but the following information about caching in the gateway environment should be noted.

The gateway server MUST follow all Cache-Control directives when caching content from content servers. However, a gateway server MAY interpret the Cache-Control "private" mode to mean that the content is cached for a particular user of the gateway server, and the gateway server consequently MAY cache private information.

6.4. Redirection

HTTP Redirection occurs as a result of a 3XX response status code and brings up a few interesting considerations for the gateway server. Redirection will differ according to the aggregation type of the response.

6.4.1. Redirection with Single or Hosted Responses

The main question in this situation is whether a redirect from the content server will result in the page being served up by the gateway server, or the client being redirected directly to the redirect URI. Both of these strategies are useful, and each should be used at different times. This specification does not specify how a gateway server will determine which URIs will be “gatewayed” versus which URIs are outside the "Gateway Space." A server COULD act as a gateway for any URI on the content server, it COULD only gateway a predetermined list of URIs, or it COULD never gateway any URI except the initial piece of content it requested. Gateway configuration is outside the scope of this protocol. The implementer of the gateway server should carefully consider the security implications for whichever strategy is chosen.

Dealing with a redirection through the gateway can be done in one of two ways. The gateway server can either follow the redirect itself, or it can send back a 3XX response to the client with a Location header that is translated into the gateway space. For this reason, the gateway server MAY modify a Location Response header.

Note: If the gateway server sends a 3XX response to a client that sends the client directly to a new origin server (including sending the client directly to the content server), the new origin server will not receive any of the headers in this specification. Any cookie headers sent back from the new origin server will be stored on the client user agent, not the gateway server. For all intents and purposes, a 3XX response with a new origin server makes the communication leave the purview of this specification.

6.4.2. Redirection with Multiple Responses

When a content server resource is being aggregated by the gateway server (i.e. the aggregation type is Multiple), the redirect header cannot be sent back to the client. In this case, the gateway server MUST either follow redirects by itself or return an error message to the client. In the Multiple aggregation type case, the gateway server MUST NOT pass through an HTTP redirect to the client.

6.5. Ranges

Because of content modification (section 5.2) and aggregation of multiple HTTP responses, supporting the HTTP 1.1 notion of ranges could be very difficult for a gateway server. This is because byte ranges that the client knows about will correspond to different byte ranges on the content server. For example, a byte range sent by a client for an aggregated response could correspond to pieces of none, one, or multiple content server HTTP responses. It seems that all but the most complicated implementations would need to reconstruct the entire aggregated HTTP response in order to satisfy a range request, thus losing most of the performance gain of range requests.

For this reason, a gateway server MAY choose not to implement HTTP 1.1 ranges. This means that a gateway server MAY remove any Range request header when passing on a request to a content server. A gateway server MAY also remove any Accept-Ranges response header or MAY replace any Accept-Ranges response header with an "Accept-Ranges: none" header.

6.6. Separate HTTP Headers Will Not Be Sent

Not all HTTP headers from a content server will be passed through to the client. The gateway server MAY base its response headers on the headers from the content server, but a content server SHOULD NOT expect any of its HTTP headers to passed through to the client. An example of this is the Last-Modified header. The gateway server COULD choose the latest last modified date of the aggregated HTTP responses and return that date in the Last-Modified header, it COULD use some other algorithm not based on the content server responses to determine a last modified date, or it COULD not send back a Last-Modified header at all.

6.7. HTTP Header Modification

Note: This section only applies to those requests that the gateway server classifies as aggregation type "Single" or "Hosted." (See section 5.1.) The gateway server will always construct its own set of HTTP response headers for any request of aggregation type "Multiple." It MAY base those headers on the headers from the content server(s), but content servers SHOULD NOT expect any of their HTTP response headers to passed through to the client.

As part of being a gateway for un-aggregated content, the gateway server MAY modify various HTTP headers from the content server. This section describes which HTTP headers from RFC 2616 and RFC 2109 will be passed through to the client unmodified, which ones will be deleted, and which ones will be changed. Gateway server behavior is unspecified for headers that are not defined by RFC 2616, RFC 2109, or this specification, but the gateway server SHOULD pass those headers through to the client unless it has reason to do otherwise.

6.7.1. Headers That Must Not Be Modified

The gateway server MUST NOT modify any of the following HTTP headers:

- Accept
- Accept-Charset
- Age
- Allow
- Content-Encoding
- Content-Language
- Content-Type
- Etag
- Expires
- From
- If-Match
- If-Modified-Since
- If-None-Match
- If-Unmodified-Since
- Last-Modified
- Referer
- Retry-After
- User-Agent
- Vary

6.7.2. Headers That May Be Changed, Translated, or Deleted

The following headers MAY be modified by the gateway server, either as a result of content modification/aggregation or in order to be a gateway that is compliant with RFC 2616.

6.7.2.1. Accept-Encoding

If the gateway server does not support the content encoding used by the client, it MAY remove the Accept-Encoding header.

6.7.2.2. Accept-Language

The gateway server MAY modify the Accept-Language header to contain the language (locale) selected by the user on the gateway server.

6.7.2.3. Accept-Ranges

If the gateway server does not support ranges, it MAY remove the Accept-Ranges response header. Alternately, it MAY replace any Accept-Ranges value with "none."

6.7.2.4. Authorization

The gateway server MAY replace the Authorization header or add a new Authorization header with credentials stored on the gateway server.

6.7.2.5. Cache-Control

The gateway server includes a `Cache-Control : no-cache` header in every request to a content server to prevent caching of requests by proxy servers.

6.7.2.6. Connection

The gateway server MUST NOT forward the Connection header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.7. Content-Length

If the gateway server modifies the content returned, it MUST modify the Content-Length header to reflect the new Content-Length. Otherwise, the gateway server SHOULD pass through the Content-Length returned by the content server.

6.7.2.8. Content-Location

The gateway server MAY modify the Content-Location field to a URI which corresponds to the Content-Location URI obtained through the gateway.

6.7.2.9. Content-MD5

If the gateway server modifies the body of the content server response, the gateway server MUST delete this header. This is because the body modification will cause the MD5 digest to be invalid, and the gateway server is specifically prohibited from generating this header by RFC 2616 (section 14.15). If the gateway server does not modify the body of the content server response, the gateway server MAY pass the Content-MD5 header on to the client.

6.7.2.10. Content-Range

If the gateway server does not support ranges, it should never receive this header from a content server. If it does support ranges, it MAY translate the Content-Range header to accommodate the effects of content modification and aggregation.

6.7.2.11. Cookie

The gateway server MAY remove Cookie headers on the request from the client and replace them with preferences and session cookies stored on the gateway server. The gateway server MUST store cookies that do not have an expires date in a non-persistent way and send them back to the content server in the Cookie header for the duration of the logical session between the gateway server and client.

6.7.2.12. Date

The gateway server MAY modify the Date header to contain the current date on the gateway server.

6.7.2.13. Host

The gateway server MUST change the Host request header when making a request to a content server. The client will provide the hostname of the gateway server, and the gateway server MUST change it to the appropriate hostname for the content server.

6.7.2.14. If-Range

If the gateway server does not support ranges, it SHOULD remove the If-Range request header. If it does support ranges, it MAY modify the value to one that the content server will understand.

6.7.2.15. Keep-Alive

The gateway server MUST NOT forward the Keep-Alive header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.16. Location

The gateway server MAY translate the Location header to a URI in the gateway space in order to make the client redirect to another resource through the gateway.

6.7.2.17. Max-Forwards

In compliance with RFC 2616, section 14.31, the gateway server MUST decrement the value of Max-Forwards if it is greater than 0, and not forward the request if Max-Forwards is 0. Note: the Max-Forwards header is ignored if the request is not a TRACE or OPTIONS request.

6.7.2.18. Pragma

The gateway server includes a `Pragma: no-cache` header in every request to a content server to prevent caching of requests by proxy servers.

6.7.2.19. Proxy-Authenticate

The gateway server MUST NOT forward the Proxy-Authenticate header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.20. Proxy-Authorization

The gateway server MUST NOT forward the Proxy-Authorization header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.21. Range

If the gateway server does not support ranges, it SHOULD remove the Range request header. If it does support ranges, it MAY translate the Range request header to accommodate the effects of content modification and aggregation.

6.7.2.22. Set-Cookie

The gateway server MUST remove all Set-Cookie headers and not return them to the client. If the cookie has an expires date, the gateway server MAY record the cookie and send it (subject to its path, domain, and secure modifiers) in a Cookie header on further requests to the content server until it expires. If the cookie does not have an expires date, the gateway server MUST record the cookie and send it (subject to its path, domain, and secure modifiers) in a Cookie header on further requests to the content server for the duration of the logical session between the gateway server and the client.

This section does not mean that the gateway server cannot send back other Set-Cookie headers to the client for its own purposes (e.g., starting a session).

6.7.2.23. Server

The gateway server MAY replace the Server header with a string identifying the software used to run the gateway server.

6.7.2.24. TE

The gateway server MUST NOT forward the TE header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.25. Trailer

The gateway server MUST NOT forward the Trailer header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.26. Transfer-Encoding

The gateway server MUST NOT forward the Transfer-Encoding header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.27. Upgrade

The gateway server MUST NOT forward the Upgrade header, because it is a hop-by-hop header, as defined in RFC 2616, section 13.5.1.

6.7.2.28. Via

The gateway server MUST add itself to the Via header on both requests and responses in compliance with RFC 2616, section 14.45.

APPENDIX A. Enhancements and Backwards Compatibility (1.3 – 1.4)

The 1.4 version of CSP is a superset of the 1.3 version of the protocol. All content servers designed to work with the 1.3 protocol should work correctly with 1.4. Version 1.4 includes a new CSP-Gateway-Config value called PT-Identifier that provides the name of the current object (portlet, pagelet, page, etc.).

APPENDIX B. Enhancements and Backwards Compatibility (1.2 – 1.3)

The 1.3 version of CSP is a superset of the 1.2 version of the protocol. All content servers designed to work with the 1.2 protocol should work correctly with 1.3. Version 1.3 includes a new type of preferences (session-level), new request and response headers, and new CSP-Gateway-Specific-Config values.

APPENDIX C. Enhancements and Backwards Compatibility (1.1 – 1.2)

The 1.2 version of CSP is a superset of the 1.1 version of the protocol, except that functionality deprecated in 1.1 has been removed entirely in 1.2. All content servers designed to work with the 1.1 protocol that did not use features deprecated in 1.1 should work correctly with 1.2. Version 1.2 includes new request and response headers, new CSP-Gateway-Specific-Config values, and some modifications to the requirements for handling Cookie and Set-Cookie headers related to the removal of deprecated 1.0 functionality.

APPENDIX D. Enhancements and Backwards Compatibility (1.0 – 1.1)

The 1.1 version of CSP is a superset of the 1.0 version of the protocol. All content servers designed to work with the 1.0 protocol should work correctly with 1.1. There are some basic changes in functionality, detailed below.

The 1.1 protocol supports “pathless” setting of all preferences. The “pathful” aspects of the 1.0 protocol with regard to preferences will be deprecated in 1.1. In 1.0, pathfulness was enforced through the use of cookies. “Set-Cookie” headers are associated (explicitly or implicitly) with a path and a domain, and clients that receive those headers are expected to respect the path and domain (i.e., to only send cookies back to the correct resources in the correct servers). Since preferences don’t have paths or expiration dates as cookies do, they are significantly easier to work with and more predictable.

Cookies and Preferences

In version 1.0 of CSP, Global-Gadget, Gadget-Realm-User and Gadget-Realm level preferences were sent as name-value pairs in a standard HTTP Cookie header, instead of in a proprietary header.

1.1 provides explicit support for ALL preferences via portal-specific headers. Since cookies are not required in 1.1, the content server should return the portal-specific header with the name=value syntax directly in the header. (See the examples in section 3.)

In 1.0, certain preferences were passed using the standard Set-Cookie header.

- Global-Gadget and Gadget-Realm level preferences were set by returning a portal-specific header that specified a cookie name, and a “Set-Cookie” header that contained the preference name and value.

- Gadget-Realm-User level preferences corresponded directly to cookies; the protocol specified no separate mechanism for setting or returning these preferences (i.e. if the content server returned a persistent cookie, and the cookie name was not mentioned in any Global-Gadget or Gadget-Realm header, it was assumed to be a Gadget-Realm-User level preference).

For backwards compatibility, setting these preferences through cookies will be supported in 1.1. However, the preferred method of setting preferences is within portal-specific headers, and 1.0 functionality will probably not be supported in future versions.

Note: Cookies are supported in the 1.1 protocol, and will NOT be deprecated. In the 1.1 protocol, cookies are just cookies, and follow normal cookie rules. Content servers that wish to set Gadget-Realm-User level preferences should use the appropriate portal-specific header.

“Set” Headers

In 1.1, explicit “Set” style headers are supported for ALL preference types. The 1.0 protocol defined portal-specific headers of two different formats.

- Global-Gadget and Gadget-Realm level preferences used headers of the form CSP-[pref type]-Pref to both set and get Preferences.
- User and Community level preferences used an explicit CSP-Set-[pref type]-Pref header to set preferences, and CSP-[pref type]-Pref to return them.

In 1.1, all preferences have an associated CSP-Set-[pref-type]-Pref header that should be used by the content server to set preferences on the gateway server. For backwards compatibility, headers of the format CSP-[pref type]-Pref, will be considered equivalent to the appropriate “Set” header.