Creating CSP-Based Web Applications

Version 2023.3
2024-05-16

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Creating CSP-Based Web Applications
InterSystems IRIS Data Platform Version 2023.3 2024-05-16

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

Introduction to CSP-Based Web Applications

InterSystems IRIS® data platform provides a technology that enables you to create web user interfaces. For historical reasons, this technology is known as *CSP* and the result is a *CSP-based web application*.

CSP intercepts HTTP requests before the browser sees them, and CSP provides an easy way to write directly to the browser. These factors mean that you have complete control over the web pages you generate. You can use CSP in conjunction with third-party JavaScript libraries if wanted, because you can simply add any needed JavaScript to your pages.

1.1 Components of a CSP-Based Web Application

From the point of view of an application developer, a CSP-based web application consists of the following elements on an InterSystems IRIS server:

- A web application definition that is configured to use **CSP/ZEN** options instead of **REST** options.

  The web application definition controls things such as the allowed authentication mechanisms, the InterSystems IRIS namespace to run the code in, custom pages to use for specific purposes, timeouts, handling of static files, and more.

- One or more CSP page classes. These classes generate full HTML pages in response to HTTP requests. A CSP page class has easy access to the request and to session information, and utility methods enable you to manage cookies, access CGI variables, and more.

- External JavaScript files, style sheets, image files, static HTML files, and other resources as needed. These can live on the same server, can be on other servers, or can be a mix of both.

Architecturally, there are additional components:

- A web server, as supported for use with InterSystems IRIS.

- The *Web Gateway*, which acts as the intermediary between the web server and the InterSystems IRIS server or servers. The Web Gateway maintains a pool of connections to these servers.

  The Web Gateway configuration includes definitions of web applications running on the InterSystems IRIS servers. These definitions enable the Web Gateway to route requests to the correct web applications on the InterSystems IRIS servers (sending the requests via the CSP server, discussed next).

- The *CSP Server*, which is a dedicated process running on an InterSystems IRIS server that waits for requests from the Web Gateway and then handles them as needed. Each InterSystems IRIS server may run as many CSP server processes as desired (subject to limits imposed by machine type; CSP servers are not counted in license calculations).

1.2 Information Flow

For a CSP-based web application, the request and return process is as follows:

1. An HTTP client, typically a web browser, requests a page from a web server using HTTP.
2. Because of how web server is configured, it recognizes the request as a CSP request and forwards it to the Web Gateway.
3. The Web Gateway determines the InterSystems IRIS server to talk to and forwards requests to the CSP server on that target system.
4. The CSP Server processes the request and determines whether the request is for a static file or for a CSP class.

   The CSP Server includes the *Stream Server*, which is the class %CSP.StreamServer. The Stream Server is responsible for handling files and other streams. If the request is for a static file, the Stream Server finds the file within the local file system, determines how to encode it, and packages it.

   If the request is for a class, the CSP server calls the **Page()** method of the class, which in turn calls the callback methods defined in that class.
5. The CSP Server returns the requested content to the Web Gateway, which passes it back to the web server.
6. The web server sends the content to the browser for display.

2

Configuring a CSP-Based Web Application

You can use the same Management Portal pages (and the same APIs) to configure CSP-based web applications as you do for other applications. This page primarily discusses the settings that are specific to CSP-based applications.

To configure a CSP-based web application:

1. Follow the instructions in Create and Edit Applications to display the application definition (creating it first if needed).
2. For the **General** tab, use the information below to define the web application.
3. For the other parts of the application definition, see Create and Edit Applications.
4. Ensure that the Web Gateway configuration is also updated to provide access to this web application. (For background, see Components of a CSP-Based Web Application.)

2.1 General Settings for a CSP-Based Web Application

For a CSP-based web application, specify values in the initial section of the **General** tab as follows:

Name

Specifies the identifier for the application. The name must include a leading slash (/), such as in the /myorg/myapp application.

Note that the name `/csp/docbook` is reserved.

Description

Specifies a text description of the application.

Namespace

Specifies the namespace where this application runs. When you select a different namespace, the dialog immediately displays the default application for that namespace to the right of this drop-down menu.

Namespace Default Application

Specifies whether the application is the default application for this namespace. The %System.CSP.GetDefaultApp() method returns the default application for the namespace. InterSystems IRIS® data platform import functions, such as $system.OBJ.Load() or $system.OBJ.ImportDir(), use the default application when importing a page without an associated application.

Enable Application

Specifies whether the application is available for use. When enabled, an application is available, subject to user authentication and authorization; when disabled, it is not available.

Enable REST or CSP/ZEN

Select **CSP/ZEN**.

Analytics

Specifies whether to enable use of Business Intelligence and Natural Language Processing within this application.

Inbound Web Services

Specifies whether to serve SOAP requests from within this application. Uncheck to disable.

Prevent login CSRF attack

Specifies whether the application automatically prevents Cross-Site Request Forgery (CSRF) attacks. InterSystems recommends that you enable this option for all new applications; it is also recommended for all existing applications except if there is code that programmatically requests pages from the application.

2.2 Security Settings

For a CSP-based web application, specify security settings as follows:

Resource Required

A resource for which users must have the Use permission so they can run the application.

Note that you can also specify required resources within CSP page classes. Both mechanisms are applied.

Group by ID

For use by By-ID groups to enable you to share authentication across multiple web applications. Enter a group name for this application to share authentication privileges with all other applications with this group name.

Allowed Authentication Methods

The application's supported authentication mechanisms, from the set of defined mechanisms.

Permitted Classes

Specify classes whose methods can be executed within this application. There are three ways to do this:

- Use an ObjectScript match pattern. Example: `1"myclass".3N` allows `myclass123.cls` to run in this application, but not `myclassxy.cls`
- Use an ObjectScript expression that evaluates to a boolean, prefixed with @. The requested class name is passed as a variable named *class*. Example: `@class = "PermittedClasses.PermittedPage"`
- Use a call to a class method (can also use @syntax). Example: `##class(MyPackage).CheckClassIsPermitted(class)`

See also Enabling Application Access to %CSP Pages.

2.3 Session Settings

For a CSP-based web application, specify **Session Settings** as follows:

Session Timeout

The default session timeout in seconds. You can override this value programmatically.

Note that a user moves from one web application to another during a session, the timeout period is still controlled by the first web application used during the session. For example, if a session starts out in Web Application A, with a default timeout of 900 seconds, and then moves into Web Application B, which has a default timeout of 1800 seconds, the session still times out after 900 seconds. In such a case, you may want to define a session event class to update the session timeout.

Event Class

The default name of the class that handles web application events, such as a timeout or session termination.

Use Cookie for Session

Whether or not the application tracks the browser session by using a cookie. Choices are:

- **Always** — *Default*. Always use a cookie to track the browser session.
- **Never** — Never use a cookie to track the browser session.
- **Autodetect** — Use a cookie to track the browser session unless the client browser has disabled them. If the user has disabled cookies, the application uses URL rewriting to track the browser session.

Note that this option does not set whether the application uses cookies; rather, it controls *how the application manages sessions*, subject to the user's preferences. Further, even with values of **Always** or **Autodetect**, an application uses cookies only if it contains specific code to do so.

Session Cookie Path

The portion of the URL that the browser uses to send the session cookie back to InterSystems IRIS for this application. If you do not specify a value for this field, the application uses the value of the **Name** field with leading and following slashes as its default scope. Hence, for an application named `myapp`, specifying no value here means that /myapp/ is the scope.

The application only sends the cookie for pages within the specified scope. If you restrict the scope to pages required by a single web application, this prevents other web applications on this machine from using this session cookie; it also prevents any other web application on this web server from seeing the cookie.

Note that a primary application and its subapplications can have different security settings while simultaneously sharing a session cookie (if they all use the primary application's path).

Session Cookie Scope

Controls the default value of the SameSite attribute for session cookies.

User Cookie Scope

Controls the default value of the SameSite attribute for application-specific cookies.

2.3.1 About the SameSite Attribute

The SameSite attribute determines how an application handles cookies in relation to third-party applications (aka *cross-site requests*). SameSite can have a value of:

- **None** — The application sends cookies with cross-site requests. If SameSite has a value of **None**, browsers may require applications to use HTTPS connections.
- **Lax** — The application sends cookies with safe, top-level cross-site navigation.
- **Strict** — The application does not send cookies with cross-site requests. (The default for system web applications and new or upgraded user applications.)

You can programmatically override this attribute within the CSP page classes.

The SameSite attribute is part of an initiative from the IETF and is addressed in several of their documents.

2.4 CSP File Settings

The CSP server can serve static files in addition to passing back content generated by the CSP pages. For a CSP-based web application, the **CSP File Settings** control how the CSP server handles static files. Also see How the CSP Server Handles Static Files.

You can use a traditional configuration of serving static pages from the web server. (In this case, the settings described here are irrelevant.) This may be preferable in certain situations. For example, if you have a system where one web server serves multiple remote InterSystems IRIS instances, it may be more efficient to serve those files from a common location local to the web server machine.

Note, however, that configuring the web server to serve static files may cause problems: for example, if the common web server serves different versions of InterSystems IRIS, the web server may encounter a conflict between two different versions of the same file (for example, hyperevent broker components). Such conflicts do not occur when each CSP server handles static content for its own applications. Additionally, if you have configured the web server itself to serve static files, you must make sure that the static content is present on every single web server in your system.

Serve Files

Controls whether to serve static files from the directory specified by **Physical Path**.

- **No** — Never serve files from this application path.
- **Always** — *Default*. Always serve files from this application path and ignore the CSP security setting for this path for static files. This is the default for new applications; it is backward compatible with applications that previously had static files served from the web server.
- **Always and cached** — Always serve files from this application path and allow the Web Gateway to cache these files to avoid having to request them from InterSystems IRIS. This is the mode that deployed applications are expected to use.
- **Use CSP Security** — If you have permission to view a CSP page in this application, then you can also view static files. If you do not have permission to view a page, then you see a `404 page not found` message.

Serve Files Timeout

Specifies the length of time for which the browser should cache static files (in seconds). Default is 3600.

Physical Path

The directory on the InterSystems IRIS server from which to serve files for this web application. The path is relative to the *install-dir*/csp/ directory.

Ignore the options **Package Name**, **Default Superclass**, **Recurse**, **Auto Compile**, and **Lock CSP Name**, which apply only to tag-based development. (If needed, see Editing a CSP Application: The General Tab in the Caché/Ensemble documentation.)

2.5 Custom Pages

For a CSP-based web application, specify **Custom Pages** as follows:

Login Page

Optionally specifies the name of a CSP page class, which may be prefixed with the full web application path. For example: `/csp/user/MyApp.LoginPage.cls`

Usually, the login page is loaded before the user has logged in to InterSystems IRIS, so the requesting process runs under the **`CSPSystem`** user (or whatever user connects the CSP Gateway to InterSystems IRIS). As a result, the **`CSPSystem`** user must have sufficient privileges to load and run the code in the login page, which generally requires READ permissions on the resource protecting the database in which the login page is located.

Change Password Page

Optionally specifies the name of the page class to use when changing password.

Custom Error Page

Optionally specifies the name of a CSP error page class to display if an error occurs when generating a page within this application.

2.6 Enabling Access to Pages and Classes

The following rules govern access to pages and classes from within a web application:

1. By default, a user application is allowed to access the following pages:
   - Pages of the /csp/sys/ application and all of its subapplications are allowed.
   - Pages of the /isc/studio/templates/ and /isc/studio/usertemplates/ applications are allowed.
2. By default, a user application is allowed to access all non-% classes in the current namespace.
3. A user application can also access the following classes:
   - %CSP.Broker, %CSP.StreamServer, %CSP.Login, %CSP.PasswordChange, %CSP.PageLookup are allowed.
   - %ZEN.SVGComponent.svgPage and %ZEN.Dialog.* are allowed, with the following additional conditions:
     - All other %ZEN.* classes are *not* allowed.
     - All other %Z* classes *are* allowed.
   - All %z* classes are allowed.

Checking for allowed classes is performed in addition to checking the setting in the web application.

To permit access to additional classes, configure the global ^SYS("Security","CSP","category") in the %SYS namespace, where *category* is AllowClass, AllowPrefix, or AllowPercent. The following sections describe these options.

Important: Checking is done by applying the default rules first, then the categories in the order listed.

Also, each keyword can be invoked more than once. This means that you can make an entire package accessible, and then restrict access to one class in that package.

2.6.1 Background Information on the ^SYS Global

The ^SYS global is available in the %SYS namespace and contains configuration information. You may find it helpful to start by examining the current contents of the relevant part of this global. To do so, open the Terminal and switch to the %SYS namespace. Then enter the following command:

ObjectScript

```
zw ^SYS("Security", "CSP")
```

The system then displays one line for each node, showing its current value. For example:

```
^SYS("Security","CSP")=1
^SYS("Security","CSP","AllowClass","/csp/samples/","%CSP.UI.Portal.About")=1
^SYS("Security","CSP","AllowClass","/csp/samples/","%SOAP.WebServiceInfo")=1
^SYS("Security","CSP","AllowClass","/csp/samples/","%SOAP.WebServiceInvoke")=1
^SYS("Security","CSP","AllowPrefix","/csp/samples/","%DeepSee.")=1
```

2.6.2 Category: AllowClass

If your application relies on invoking a particular class, use the AllowClass option to make that class available.

Important: If your application relies on invoking any class other than those listed as *allowed* at the beginning of Enabling Application Access to %CSP Pages, it could potentially be unsafe to use. InterSystems recommends that you determine if calling this class is required, and perform a risk assessment for your deployment, so that you understand the implications of making the class available.

To enable a given web application to invoke a particular class, use the following command in the %SYS namespace:

```
Set ^SYS("Security", "CSP", "AllowClass", "web-app-name", "package.class") = value
```

Where:

- *web-app-name* is the name of the web application. The web application name must be in lowercase and must start *and* end with a trailing slash.

  To enable *all* web applications to use the given class or package, specify *web-app-name* as 0; in this case, you can omit the enclosing quotes.

- *package.class* is the fully qualified name of a class. If you omit *class*, then *all* classes in the specified package are allowed.

- *value* is either 1 or 0.

  If you specify this as 1, the web application can invoke this class (or package).

  If you specify this as 0, this web application cannot invoke this class (or package).

For example, to enable the /csp/webapps application to use the class %User.Page, you would use the following command:

```
Set ^SYS("Security", "CSP", "AllowClass", "/csp/webapps/", "%User.Page") = 1
```

Or to enable *all* web applications to use the %User.Page, you would use the following command:

```
Set ^SYS("Security", "CSP", "AllowClass", 0, "%User.Page") = 1
```

For another example, to enable the /csp/myapp application to use all classes in the %User package except for the %User.Other class, you would use the following two commands:

```
Set ^SYS("Security", "CSP", "AllowClass", "/csp/myapp/", "%User") = 1
Set ^SYS("Security", "CSP", "AllowClass", "/csp/myapp/", "%User.Other") = 0
```

2.6.3 Category: AllowPrefix

If your application relies on invoking multiple classes or packages that begin with the same set of characters, use the `AllowPrefix` option.

Important: If your application relies on invoking any class other than those listed above, it could potentially be unsafe to use. InterSystems recommends that you determine if calling this class is required, and perform a risk assessment for your deployment, so that you understand the implications of making the class available.

To enable a given web application to invoke classes or packages that begin with the same set of characters, use the following command in the %SYS namespace:

```
Set ^SYS("Security", "CSP", "AllowPrefix", "web-app-name", "prefix") = value
```

Where:

- *web-app-name* is the name of the web application. The web application name must be in lowercase and must start *and* end with a trailing slash.

  To enable *all* web applications to use the given classes or packages, specify *web-app-name* as 0; in this case, you can omit the enclosing quotes.

- *prefix* is the first characters in the name.

- *value* is either 1 or 0.

  If you specify this as 1, the web application can invoke these classes (or packages).

  If you specify this as 0, this web application cannot invoke these classes (or packages).

For example, to enable the /csp/webapps application to invoke the entire MyApp package, use the following command:

```
Set ^SYS("Security", "CSP", "AllowPrefix", "/csp/webapps/", "MyApp.") = 1
```

Note that *prefix* is `"MyApp."` and the period in the prefix means that the web application cannot access the packages such as `MyAppUtils`. The web application can, however, access the packages `MyApp.Utils` and `MyApp.UnitTests`.

For another example, to enable *all* applications to access *all* packages that begin with `My`, use the following command:

```
Set ^SYS("Security", "CSP", "AllowPrefix", 0, "My") = 1
```

For another example, suppose that the /csp/myapp application should be able to access all classes in the %MyPkg package except for the class %MyPkg.Class1. In that case you would use the following two commands:

```
Set ^SYS("Security", "CSP", "AllowClass", "/csp/myapp/", "%MyPkg.Class1") = 0
Set ^SYS("Security", "CSP", "AllowPrefix", "/csp/myapp/", "%MyPkg.") = 1
```

2.6.4 Category: AllowPercent

If your application relies on invoking the packages that begin with the % character generally, the `AllowPercent` option makes those classes available.

Important: If your application relies on invoking any class other than those listed above, it could potentially be unsafe to use. InterSystems recommends that you determine if calling this class is required, and perform a risk assessment for your deployment, so that you understand the implications of making the class available.

To enable all web applications to use all packages that begin with the % character, use the following command in the %SYS namespace:

```
Set ^SYS("Security", "CSP", "AllowPercent") = 1
```

Note: Or use the value 0 to explicitly forbid any web application from accessing these packages.

2.7 How the CSP Server Handles Static Files

For a web application that is configured to serve static files, the CSP server — specifically its Stream Server component — processes static files. It uses the file extension to determine:

- The file type (the MIME type)
- Whether the file is a binary file
- The character encoding of the file (if applicable)

For JavaScript files, the Stream Server determines the character encoding in a manner consistent with major web servers, described below. You can override the default behavior if needed.

2.7.1 Character Encoding of JavaScript Files

The modern convention is for all JavaScript files to be marked as Content-Type of `application/javascript`. With JavaScript files marked this way:

- If a file contains a BOM (byte-order mark), the browser automatically detects this and uses the correct character set to read it.
- If the file does not contain a BOM, then the browser assumes the file is UTF-8.

If you need to override this behavior to specify a character set for JavaScript files, set the global *^%SYS("CSP","MimeFileClassify","JS")* to the list value **$listbuild(contenttype, binary, charset)**. For example,

```
Set list=$listbuild("text/javascript", 0 ,"ISO-8859-1")
Set ^%SYS("CSP", "MimeFileClassify", "JS") = list
```

This sets the older content-type and uses the ISO-8859-1 character set. Also, if the default InterSystems IRIS translate table is defined to be something other than an empty string or if the global node *^%SYS("CSP","DefaultFileCharset")* is set to a null value, InterSystems IRIS will use this character set for all JavaScript and other text files. By default, neither of these global nodes are set.

3

Creating a CSP Page Class

A CSP page class processes requests and generates HTML, writing that HTML directly to the browser. When you create a CSP-based web application, your primary task is to define the **OnPage()** callback method of your page class or classes.

3.1 Basics

To create a CSP page class, do the following:

1. Create a subclass of %CSP.Page.
2. In this subclass, implement the **OnPage()** callback method. This method has access to the following variables:
   - *%request*, which contains properties with information about the request. This variable is an instance of %CSP.Request.
   - *%session*, which contains information about the browser session. This variable is an instance of %CSP.Session.
   - *%response*, which contains information about the default HTTP response. This variable is an instance of %CSP.Response.

   Your method should examine that information as needed, generate the HTML page as a string, and use the **Write** command to write that string. InterSystems IRIS® data platform automatically redirects the standard output device (*$IO*) so that all **Write** output is sent back to the HTTP client.

   %CSP.Page provides helpful class methods you can use to escape and unescape strings for use in HTML and JavaScript contexts.

   Also see Special HTML Directives for an alternative approach.
3. Optionally override parameters of %CSP.Page, which enable you to control:
   - Default response headers sent to the client
   - Access to the page
   - Encryption
   - Error pages displayed in specific scenarios

   For a list of all available class parameters, refer to the documentation for %CSP.Page.
4. Optionally override other callback methods to control processing at additional times.

For example:

Class Definition

```
Class GCSP.Basic Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   Set html="<!DOCTYPE html>"
           _"<html lang=""en"" dir=""ltr"">"
           _"<body>"
           _"<h1>Basic Page</h1>"

   // examine %request object and
   // write more output...

   Set html=html_"</body>"
               _"</html>"

   Write html //finally, write the page
   Quit $$$OK
}

}
```

This example concatenates the entire page as a single string and then writes it, which is fine for pages expected to be under the long string limit. The following variation produces the identical HTML:

Class Definition

```
Class GCSP.Basic Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   write "<!DOCTYPE html>"
           _"<html lang=""en"" dir=""ltr"">"
           _"<body>"
           _"<h1>Basic Page</h1>"

   // examine %request object and
   // write some more html...

   write "</body>"
         _"</html>"

   Quit $$$OK
}

}
```

Note that when there are multiple pages that need to have the same overall appearance, it is better to have separately testable helper methods for constructing the page start, the <head> with the HTML meta data, any JavaScript, links to style sheets, the navigation s, the footer, the page end, and so on.

3.2 Controlling the Default Response Headers

Class parameters in your CSP page class determine the default HTTP response headers sent to the client.

To control the type of content returned to the browser, specify the *CONTENTTYPE* class parameter. For example:

Class Member

```
Parameter CONTENTTYPE = "application/vnd.ms-excel";
```

Similarly, to control the character set used, specify the *CHARSET* class parameter:

Class Member

```
Parameter CHARSET = "UTF-8";
```

For a list of all available class parameters, refer to the documentation for %CSP.Page.

You can also control the HTTP headers of the response by setting properties of the *%response* object, within the **OnPreHTTP()** callback, discussed next.

3.3 Other Callbacks

When InterSystems IRIS® data platform determines which %CSP.Page class should process a request, it calls the **Page()** method of the class, which in turn calls these callback methods in order:

1. **OnPreHTTP()**
2. **OnPage()** discussed above
3. **OnPostHTTP()**

3.3.1 OnPreHTTP()

You can implement **OnPreHTTP()** for finer control of the HTTP headers of the response.

Specifically, you can set properties of the *%response* object, such as the ContentType property, if that needs to be different from the default that you chose:

```
set %response.ContentType="text/html"
```

Then InterSystems IRIS uses those values when writing to the client.

The **OnPreHTTP()** method is also where you can define redirects, if needed. You can create a normal client-side redirect by setting the *Redirect* property:

```
set %response.Redirect="https://someotherurl"
```

Or, if you want to invoke another CSP page, you can instead set the *ServerSideRedirect* property, which will cause the CSP Server to invoke the **Page()** method in the specified class.

```
set %response.Redirect="GCSP.OtherPage.cls"
```

Note that a server-side redirect does not change the URL seen in the browser.

3.3.2 OnPostHTTP()

The **OnPostHTTP()** method is provided as a place to perform any operations you wish to perform after processing of the HTTP request is complete.

3.4 Best Practice for Links

When you include an HTML <A> anchor link, use the **Link()** method of %CSP.Page to create the URL. This method performs any URL escaping and also encrypts the URL parameters if applicable (depending on the definition of the target page).

This method has the following signature:

```
classmethod Link(link As %String, ByRef query As %String,
                 addQ As %Boolean = 0) as %String
```

Where:

- *link* is the base URL
- *query* is a multidimensional array containing any URL parameters
- *addQ* is a Boolean value specifying whether to include a trailing ? or & (as appropriate) at the end of the returned value. This option enables you to append additional query parameters.

For example:

ObjectScript

```
Set origurl="GCSP.EncryptPage2.cls"
Set urlparms("SAMPLEPARM")="sample value"
Set tURL = ##class(%CSP.Page).Link(origurl,.urlparms)
Set html="<p>Link to page 2: <a href="""_tURL_""">Link</a>"_"</p>"
```

You can also use the Context property of the *%response* object to automatically insert values into all links and forms; see the class reference for details.

3.5 Tag-Based Development (Legacy)

Legacy applications may also include .csp files, which are used in *tag-based development*. In the tag-based development model, the developer creates .csp files contained within the directory structure accessed by the web application. The files contain a mix of HTML and specialized tags that provide for communication with the server. The CSP compiler reads the files and generate class definitions from them, and the class definitions generate the actual runtime HTML. For information on tag-based development, consult Tag-based Development with CSP in the Caché/Ensemble documentation.

4

Examining the Request

When the CSP server responds to an HTTP request, it packages information about the incoming request into the *%request* object, which is available on all CSP pages. This variable is an instance of %CSP.Request.

Note that the CSP pages also have access to the *%session*, which you can use to pass additional data from page to page.

4.1 URL

To find the URL (not including the query string) of an incoming HTTP request, use the URL property of the *%request* object:

```
Write "URL: ", %request.URL
```

4.2 URL Parameters

A URL may contain a list of parameters (also known as the URL query). The *%request* object makes these available via its Data property.

For example, suppose the incoming URL contains:

```
/csp/user/MyPage.csp?A=10&a=20&B=30&B=40
```

You can retrieve these parameters on the server using:

```
Write %request.Data("A",1)    // this is 10
Write %request.Data("a",1)    // this is 20
Write %request.Data("B",1)    // this is 30
Write %request.Data("B",2)    // this is 40
```

Data is a multidimensional property and each value stored within it has 2 subscripts: the name of the parameter and the index number of the parameter (parameters can occur multiple times within a URL as with B above). Note that parameter names are case-sensitive.

Also note that it does not matter if an incoming HTTP request is a GET or a POST request: the Data property represents the parameter values in exactly the same way.

You can use the ObjectScript **$Data** function to test if a given parameter value is defined:

```
If ($Data(%request.Data("parm",1))) {
}
```

If you wish to refer to a parameter but are not sure if it is defined, you can use the ObjectScript **$Get** function:

```
Write $Get(%request.Data("parm",1))
```

You can find out how many values are defined for a particular parameter name using the **Count** method of the *%request* object:

```
For i = 1:1:%request.Count("parm") {
   Write %request.Data("parm",i)
}
```

The same techniques also work even when the parameter values are encrypted.

4.3 Form Data

When the CSP server receives a form submit request, it places all name/value pairs into the multidimensional Data property of the *%request* object, where the subscripts of Data are the names.

For example, if the form looks like this:

```
<form name="edit" method="post" action="GCSP.EncryptPage3.cls">
<p>Value A: <input type="text" name="inputA"></p>
<p>Value B: <input type="text" name="inputB"></p>
<p><input type="submit" value="Submit This" ></p>
</form>
```

Then on the server, use this to get the values of the Value A and Value B inputs:

```
set myvalueA=$GET(%request.Data("inputA",1))
set myvalueB=$GET(%request.Data("inputB",1))
```

If InterSystems IRIS receives a value that is longer than the string length limit, it automatically creates a stream (an instance of %CSP.CharacterStream), writes the value to that stream, and places the stream OREF into the Data property in place of the actual value. This means that in any case where you might receive a very long string, your code should examine the value to see if it is an OREF, and then handle it accordingly:

ObjectScript

```
Set value=%request.Data("fieldname",1)
If $isobject(value) {
  ; Treat this as a stream
} Else {
  ; Treat this as a regular string
}
```

4.4 CGI Variables

The web server provides a set of values, referred to as CGI (Common Gateway Interface) environment variables, which contain information about the HTTP client and web server. You can get access to these CGI environment values using the multidimensional property CgiEnvs of the *%request* object. You can use this in the same manner as the Data property.

For example, to determine what type of browser is making the HTTP request, look at the value of the CGI environment variable *HTTP_USER_AGENT*:

```
Write %request.CgiEnvs("HTTP_USER_AGENT")
```

For information on the available CGI environment variables, see CGI Environment Variables.

4.5 MIME Data

An incoming request may contain MIME (Multipurpose Internet Mail Extensions) data. This is typically used for larger pieces of information, such as files. You can retrieve MIME data using the using the multidimensional property MimeData of the *%request* object.

5

Managing Sessions

A *session* represents a series of requests from a particular client to a particular application over a certain period of time. InterSystems IRIS® data platform provides session tracking automatically. Within your CSP-based web application, you can access information about the current session by examining the *%session* object, which is an instance of %CSP.Session.

For information on sharing authentication sessions or data among applications, see Authentication Sharing Strategies. Also see Ending Sessions.

5.1 Session Creation

A session starts when an HTTP client makes its first request to a web application. The InterSystems IRIS server then does the following:

1. Creates a new session ID number.
2. Performs licensing checks, as appropriate.
3. Creates the *%session* object (which is persistent).
4. Calls the **OnStartSession()** method of the session event class (if present).
5. Creates a session cookie in order to track subsequent requests from the HTTP client during the course of the session. If the client browser has disabled cookies, InterSystems IRIS automatically uses URL rewriting (placing a special value into every URL) in order to track sessions.

5.2 Basic Properties

For the first request of a session, the NewSession property of the *%session* object is set to 1. For all subsequent requests it is set to 0:

```
If (%session.NewSession = 1) {
   // this is a new session
}
```

The SessionId property of the *%session* object contains the unique identifier of the session.

5.3 Managing Session Data

The *%session* object provides an easy way to save application-specific data across the session without using cookies or URL parameters. In particular, the Data property of this object is a multidimensional array property for use by your code. For example, suppose that you want to save the following set of key-value pairs:

Key	Value
product	`"widgets"`
quantity	100
unitofmeasure	`"cases"`

To do this, you can save values in the Data property as follows:

ObjectScript

```
set %session.Data("product")="widgets"
set %session.Data("quantity")=100
set %session.Data("unitofmeasure")="cases"
```

Similarly, perhaps on a different page of the application, you can retrieve the values:

ObjectScript

```
set productodisplay=$GET(%session.Data("product"))
set quantityodisplay=$GET(%session.Data("quantity"))
set uomtodisplay=$GET(%session.Data("unitofmeasure"))
```

Note that this example uses $GET to prevent an error in the case when a particular subscript is not defined. It is best practice to use $GET (or $DATA) to protect against such an error.

Also note that this technique allows you to store only literal data (not object references) and each value in the array must be shorter than the long string limit.

5.4 Deleting Session Data

To remove data from the **Data** property, use the ObjectScript **Kill** command:

```
Kill %session.Data("MyData")
```

5.5 Customizing Session Handling (Event Class)

To customize what happens when various session events occur:

1. Define a session event class, which must be a subclass of %CSP.SessionEvents.
2. In that class, implement the callback or callbacks to customize what the application does when the session starts, times out, or when other events occur.

   The callback methods include the following:

- **OnStartSession()** — controls what happens when a session starts.
- **OnSessionEnd()** — controls what happens when a session ends. Called for all session endings.
- **OnTimeout()** — controls what happens when a session timeout occurs. Called only in case of timeout.
- **OnApplicationChange()** — controls what happens when the user moves from one application to another within a session. (In such a case, you may want to update the session timeout value).

For details, see %CSP.SessionEvents.

3. Configure the web application to use this session event class.

   Or, within your application code, programmatically specify the session event class by setting the EventClass property of the *%session* object.

5.6 Preserving Context

By default, only the processing context preserved by the CSP server from one request to the next is held within the *%session* object. The CSP server provides a mechanism for preserving the entire processing context variables, instantiated objects, database locks, open devices between requests. This is referred to as *context preserving mode*. You can turn context preservation on or off within a CSP application at any time by setting the value of the Preserve property of the *%session* object. Note that tying a process to one session results in a lack of scalability and that it is very uncommon to use this option.

6

Ending Sessions

Within an InterSystems IRIS® data platform CSP-based web application, a session can end because the user logs out, because the server ends the session explicitly, or because the session times out.

6.1 Provide Logout Option

The standard practice is to provide a link or a button with which the user can log out.

The recommended practice is to define this link or button so that it links to the application home page and to include `IrisLogout=end` in the link URL. This server then ends the current session before it attempts to run the home page.

6.2 Have the Server End the Session

From within the application, you can end a session explicitly, in the following ways:

- End the session (for example, if the client is stopped or navigates to a new site):

  ObjectScript

  ```
  set %session.EndSession=1
  ```

- Log the user out:

  ObjectScript

  ```
  do %session.Logout()
  ```

These techniques use the *%session* object that is available on the server; this is an instance of %CSP.Session.

6.3 Session Timeout

In *session timeout*, a session ends because it did not receive any requests within the specified session timeout period.

By default, the session timeout is set to 900 seconds (15 minutes). This is controlled by the web application definition.

6.3.1 Modifying the Timeout Programmatically

From within the application, you can modify the timeout by setting the AppTimeout property of the *%session* object. For example:

ObjectScript

```
Set %session.AppTimeout = 3600 // set timeout to 1 hour
```

To disable session timeouts, set the timeout value to 0.

Note that if a session changes web applications during its life span, its timeout value will not be updated according to the default timeout defined in the application that the session moved into. For example, if a session starts out in web application A, with a default timeout of 900 seconds, and then moves into web application B, which has a default timeout of 1800 seconds, the session will still timeout after 900 seconds.

If you want an application change to result in the session timeout being updated to that of the new application, define a session event class. In that class, override the **OnApplicationChange()** callback method, and add code to handle the update of the AppTimeout property of the *%session* object.

6.4 Customizing End Behavior

To customize what happens when a session ends, define a session event class and implement the **OnEndSession()** callback method of that class.

Similarly, to customize what happens when a session timeout occurs, define a session event class and implement the **OnTimeout()** callback method of that class.

6.5 Session End Details

When a session ends, the server deletes the persistent %CSP.Session object and decrements the session license count, if appropriate.

The server also deletes existing session data and removes the security context of the session.

If the session ended because of a timeout or server action, the server also calls the **OnEndSession()** method of the session event class (if it is present).

7

Saving and Using Cookies

This page describes how to save and use cookies, within a CSP-based web application.

A cookie is a name-value pair stored within the client browser. Every subsequent request from the client includes all of the previous cookie values.

Storing information within a cookie is useful for information that you want to remember past the end of a session. (To do this, you must set an expiration date as, by default, cookies end when the browser closes.) For example, you could remember a username in a cookie so that in a subsequent session they would not have to reenter this information.

7.1 Saving Cookies

To save a cookie, use the **SetCookie()** method of the *%response* as in the following example:

ObjectScript

```
Do %response.SetCookie("UserName",name)
```

A cookie definition can include an expiration date and a path in this format:

ObjectScript

```
Do %response.SetCookie("NAME","VALUE",expireData,path)
```

A blank *expireData* field defines an in-memory cookie (available only during the current session). If, however, you specify a value for the *expireData* field, this becomes a permanent cookie that is removed at the time specified. The format for the *expireData* field is `Wdy, DD-Mon-YYYY HH:MM:SS GMT`, for example: `Wednesday, 24-Mar-2024 18:12:00 GMT`.

For details, see %CSP.Response in the class reference.

7.1.1 The SameSite Attribute

When creating a cookie, you can specify the SameSite argument, which determines how an application handles cookies in relation to third-party applications (aka *cross-site requests*). This argument overrides the default *SameSite* value specified by the web application.

If you specify that a cookie has a SameSite value of **None**, then you must use an HTTPS connection.

7.2 Accessing Cookies

Any cookies are available in the Cookies property of the *%request*. This property is a multidimensional property, whose subscripts are the names of the cookies.

The *%request* object also provides methods for counting and iterating through the cookies. See **GetCookie()**, **NextCookie()**, and **CountCookie()** in %CSP.Request. For example, the following simple page class displays all cookies and their values:

Class Definition

```
Class Sample.CookieDemo Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   Set html="<!DOCTYPE html>"
            _"<html lang=""en"" dir=""ltr"">"
            _"<body>"
            _"<p>COOKIES:</p>"
            _"<ul>"

   Set cookie=%request.NextCookie("")
   While cookie'="" {
      For count=1:1:%request.CountCookie(cookie) {
         Set html=html_"<li>"_cookie_" - "
                     _..EscapeHTML(%request.GetCookie(cookie,count))
                     _"</li>"
       }
       Set cookie=%request.NextCookie(cookie)
   }
   Set html=html_"</ul>"
                _"</body>"
                _"</html>"

   Write html
   Quit $$$OK
}

}
```

8

Updating a Page without Reloading

All modern browsers have a built-in `XMLHttpRequest` object to request data from a server; with this object it is possible to interact with the server after page load and update a page without reloading it.

InterSystems IRIS® data platform provides an easy system to use this object within CSP pages, to communicate directly with the CSP server.

8.1 Basics

There are three parts to this system:

- When generating the `<head>` part of the HTML page, call **HyperEventHead()**, which returns a string consisting of two successive `<script>` elements that are required by this system. The method signature is as follows:

  ```
  classmethod HyperEventHead(iframeOnly As %Boolean,
                             strict As %Boolean = 0) as %String
  ```

  Where:

  – The *iFrameOnly* argument is ignored but is present for compatibility.

  – If *strict* is 1, the returned string will use the strict HTML 4 format of `<script>` tag.

- Create a (server-side) method to call. The method must be an instance method and can return a literal value (not an OREF). It cannot pass arguments by reference or as output.

  Tip: A common reason to `XMLHttpRequest` is to modify the page, so it may be helpful for this method to return a fully formed piece of HTML to add to the page.

- As part of the page HTML, define a JavaScript function that uses **HyperEventCall()** to invoke your method and that uses the results of that call, if applicable.

  The **HyperEventCall()** method has the following signature:

  ```
  classmethod HyperEventCall(methodName As %String,
                             args As %String,
                             type As %Integer = 0) as %String
  ```

  Where:

  – *methodName* is a reference to the server method, either in the form `..MethodName()` if the method is in the same class, or in the following longer form: `Package.Class.MethodName`

- *args* contains all the argument to pass to the method. *args* is a quoted string containing a comma-separated list of variables (which are defined earlier within your JavaScript function).
- If *type* is 1 (recommended), the call is asynchronous.

HyperEventCall() returns the value returned by the server method.

8.2 Example

An example shows how this fits together. In this scenario, we want to define a function that retrieves additional parts of a tree control. The function will use the **GetChildren()** method of a class named %CSP.Documatic.Helper. The method takes three string arguments: *name*, *parent*, and *ns* and it returns a fully formed piece of HTML that is meant to be added to the page within the tree control. (For our purposes here, it does not matter exactly how that tree control works or exactly what this method does.)

Within the code that generates the `<head>` for this HTML page, we call both **HyperEventHead()** and **HyperEventCall()** as follows:

```
 //Add hyperevent-related scripts for left navigation
 set headhtml=_..HyperEventHead()
  _"<script>function addChildrenAfter(item,name,Id,ns) {"
  _"var h="_..HyperEventCall("%CSP.Documatic.Helper.GetChildren","name,Id,ns",1)_";"
  _"if (h!==null) {"
  _"item.insertAdjacentHTML('afterend',h); } else {"
  _"location.reload();}"
  _"return false;"
  _"}</script>"
```

The resulting HTML looks like this (with line breaks added and edits for readability):

```
<script type="text/javascript" src="/somelocation/csp/broker/cspxmlhttp.js">
</script>
<script type="text/javascript" src="/somelocation/csp/broker/cspbroker.js">
</script>
<script>function addChildrenAfter(item,name,Id,ns)
{var h=cspHttpServerMethod("pyK473ekNn0...very long...DOKepQ",name,Id,ns);
if (h!==null) {item.insertAdjacentHTML('afterend',h);
 } else {location.reload();
}return false;}</script>
```

Notice that this HTML does not include the name of the server method, but instead includes a long token that the server uses to identify the code to run. Also note that this HTML includes `cspHttpServerMethod`, which is a JavaScript function provided by **HyperEventHead()**.

9

Handling Errors

For any web application, InterSystems IRIS® data platform provides a default error page that displays a message to the user if an application error occurs. You can instead provide your own custom error page. There are special options in the case when an error occurs and no license has been obtained (so server code cannot be run).

This page assumes you are familiar with ObjectScript error handling and error logging.

9.1 Adding a Custom Error Page

To add a custom error page:

1. Create a subclass of %CSP.Error and customize its **OnPage()** callback method.

   In this class, you can use the *%request*, *%response*, and *%session* objects as usual.

   In particular, *%request.Get(*`"Error:ErrorCode"`*)* contains the error information. Use **DecomposeError()** to obtain a multidimensional array containing the text of the error or errors, as follows:

   ```
   Do ..DecomposeError(%request.Get("Error:ErrorCode"),.ErrorInfo)
   ```

   Then you can loop over the *ErrorInfo* variable (returned by reference) as follows:

   ```
   For i=1:1:ErrorInfo {
        if (i=1) {
             set return="<p>"_ErrorInfo(i,"Desc")_"</p>"
        } else {
             set return=return_$CHAR(13,10)_"<p>"_ErrorInfo(i,"Desc")_"</p>"
        }
    }
   ```

   This example builds a string to be included in the HTML, with one paragraph for each error message.

   Also see %CSP.Error for additional methods for pulling information out of the error.

2. Optionally override parameters of this class to display custom pages when no license has been granted.
3. Configure the web application to use this error class.

9.2 Handling Errors Before License Grant

If a web application does not yet have a license, and a error occurs, then InterSystems IRIS displays the standard web `HTTP/1.1 404 Page Not Found` error message by default.

You can change what page is displayed when errors are encountered in such cases by specifying the following parameters in your error page class:

LICENSEERRORPAGE

This parameter controls what InterSystems IRIS does when a license cannot be granted. The parameter can have either the following two values:

- `""` or null — Returns the `HTTP/1.1 404 Page Not Found` error (default)
- Path to a static HTML file — Displays the named file, such as /csp/myapp/static.html.

PAGENOTFOUNDERRORPAGE

This parameter controls what InterSystems IRIS does if any of the following errors are generated:

- License cannot be granted
- Class does not exist
- Method does not exist
- Web application does not exist (set parameter on default error page)
- CSP page does not exist
- File does not exist
- Namespace does not exist
- Illegal request
- File cannot be opened
- Session timeout

The parameter can have either the following values:

- `""` — Return the HTTP/1.1 404 Page not found error (default)
- `1` — Obtains a license and displays the standard error page.
- Path to a static HTML file — Displays the named file, such as /csp/myapp/static.html.

OTHERSTATICERRORPAGE

This parameter controls what InterSystems IRIS does in the case of other errors. The parameter can have any the following values:

- `""` — Obtains a license and displays the standard error page (the default)
- `1` — Outputs the 404 Page not found error.
- Path to a static HTML file — Displays the named file, such as /csp/myapp/static.html.

10

Controlling Access to CSP Pages

In addition to adding authentication (not explicitly described here), you can make your pages private and you can require permissions to use pages.

These options are combined with the security settings of the web application in which the CSP pages are executed.

10.1 Making a Page Private

If you make a page private, when a user tries to display the page, the browser shows a message saying `Forbidden`.

To make a page private, so that it can be accessed only via links from other CSP pages, specify the *PRIVATE* class parameter as 1:

Class Member

```
Parameter PRIVATE = 1;
```

By default, pages are public.

10.2 Requiring Permissions to Use the Page

Use the **`SECURITYRESOURCE`** class parameter to limit access to CSP pages. For example:

Class Member

```
Parameter SECURITYRESOURCE = "%Development:USE";
```

The **`SECURITYRESOURCE`** parameter must be a comma-delimited list of system resources and the required permissions for each. You can specify an OR condition using the vertical bar (|) and an AND condition using a comma (,). A user must hold the specified permissions on all of the specified resources in order to view this page or invoke any of its server-side methods from the client.

An item in the list has the following format:

```
Resource[:Permission]
```

Resource is any of the resources defined on the server. Navigate to **System Administration** > **Security > Resources** for a list of resources.

Permission is one of USE, READ, or WRITE. Optional; default is USE.

For another example:

```
Parameter SECURITYRESOURCE = "R1,R2|R3,R3|R4" ;
```

This example means the user must have resource R1 AND one of (R2 OR R3) AND one of (R3 OR R4). If the user has R1,R3 they can run the page. If the user has R1,R4, they cannot run the page, as they do not meet the R2 OR R3 condition. The vertical bar (|) OR condition takes precedence over the comma (,) AND condition.

11

Encryption

In your CSP pages, you can encrypt values sent to the browser, including URL parameters sent to other CSP pages, by using the unique session key of the *%session*. This mechanism is secure because the session key is never sent to an HTTP client.

11.1 Encrypting and Decrypting Values

To encrypt a value, use the **Encrypt()** method of your page class. This method is inherited from %CSP.Page superclass. Within the same session, you can decrypt this value by using the **Decrypt()** method. Note that in both cases, the method automatically uses the session key.

11.2 Encrypting URL Parameters

You can encrypt URL parameters when you include an HTML <A> anchor link from one CSP page to another CSP page in the same session. The URL seen by the browser includes the *CSPToken* URL parameter instead of the original parameter or parameters (example is truncated):

```
GCSP.EncryptPage2.cls?CSPToken=1nz1Q1kNd$fJPuzngVKhsKrO...
```

There are two parts to the system:

- On the page where you are creating the URL for the link, use the **Link()** method of %CSP.Page to create the URL.

  It is best practice to use **Link()** for all URLs, whether or not they are to be encrypted.

  ObjectScript

  ```
  Set origurl="GCSP.EncryptPage2.cls"
  Set urlparms("SAMPLEPARM")="sample value"
  Set tURL = ##class(%CSP.Page).Link(origurl,.urlparms)
  Set html="<p>Link to page 2: <a href="""_tURL_""">Link</a>"_"</p>"
  ```

- On the target page, specify the *ENCODED* class parameter as either 1 or 2. This class parameter can have any of the following values:

  – ENCODED=0 — Query parameters are not encrypted. The browser receives them as is.

- ENCODED=1 — All query parameters are encrypted and passed within the *CSPToken* URL parameter (as shown in the example).
- ENCODED=2 — Same as 1 except for any unencrypted parameters, which are URL parameters appended manually to the URL (in contrast to parameters added via the **Link()** method). Unencrypted parameters are removed from the *%request*.

For example:

Class Member

```
Parameter ENCODED = 2;
```

Even when a URL parameter is encrypted as described here, you can retrieve the parameter value in the usual way. For example:

ObjectScript

```
set urlparm=$GET(%request.Data("SAMPLEPARM",1))
```

InterSystems IRIS® data platform always decrypts the value if necessary, automatically.

12

Localizing Text in a CSP Page

For general information on localizing application text, see *String Localization and Message Dictionaries*.

For a CSP-based web application, there are additional options that use the *%response* object.

12.1 Setting the Default Runtime Language

For text localized via the `$$$Text`, `$$$TextJS`, and `$$$TextHTML` macros, the default runtime language is determined by the Language property of the *%response* object, if you have set that explicitly. If the Language property is not set explicitly, the runtime language is determined by the browser settings.

The recommended way to set the Language property is to use the **MatchLanguage()** method of %Library.MessageDictionary, as in the following example:

```
Set lang = ##class(%MessageDictionary).MatchLanguage(languages,domain,flag)
Set %response.Language=lang
```

Given a list of languages and a domain name, this method uses HTTP 1.1 matching rules (RFC2616) to find the best-match language within the domain. This method has the following signature:

```
classmethod MatchLanguage(languages As %String,
                          domain As %String = "",
                          flag As %String = "") as %String
```

Where:

- *languages* is a comma-separated list of RFC1766 format language names. Each language in the list *may* be given an associated quality value which represents an estimate of the user's preference for the languages specified by the list of languages. The quality value defaults to `q=1`.

  For example, `da, en-gb;q=0.8, en;q=0.7` would mean: `I prefer Danish, but will accept British English and other types of English.` A language from the list matches a supported language tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is a hyphen (-). The special language asterisk (*), if present in the input list, matches every supported language not matched by any other language present in the list.

  The language quality factor assigned to a supported language tag is the quality value of the longest language in the list that matches the language-tag. The language that is returned is the supported language that has been assigned the highest quality factor.

- *domain* is the localization domain in which to perform the matching.

- *flag* is an optional flag indicating whether system or application messages are to be matched.

12.2 %response.GetText() Method

The *%response* object includes a **GetText()** instance method that enables you to retrieve text from the message dictionary and substitute values for any arguments the message may have.

The method signature is:

```
method GetText(language As %String = "",
               domain As %String = "",
               id As %String,
               default As %String,
               args...) returns %String
```

Where:

- *language* is an optional RFC1766 code specifying the language. InterSystems IRIS converts this string to all-lowercase. The default *language* is the Language property of the *%response*. If the Language property is not set explicitly, the language used is determined by the browser settings.
- *domain* is an optional string specifying the domain for the message. If not specified, *domain* defaults to the Domain property of the *%response* object.
- *id* is the message ID.
- *default* is the string to return if the message identified by *language*, *domain*, and *id* is not found.
- *arg1*, *arg2*, and so on are the substitution text for the message arguments. All of these are optional, so you can use **GetText()** even if the message has no arguments.

13

Authentication Sharing Strategies

This page describes how to configure multiple CSP-based web applications to work as a group in two ways:

- Sharing authentication: If applications do not share authentication, the user must log in to each application that is linked to by another application separately. Shared authentication allows the user to enter all linked applications with a single login.
- Sharing data: The applications may want to share and to coordinate global state information.

13.1 Authentication Approaches

The following approaches are available for sharing authentication.

- One-time sharing of login cookies. All applications with the same id share authentication.
- Continuous sharing by sharing sessions (having a By-Session group). In this system, the authentication of the group's applications moves as a unit. If a user in an application in a group logs in as a new user, all the applications move to that user. If one application logs out, they are all logged out.

  Applications are run in sessions. Each session has a security context associated with it.

  If several applications are placed in the same session, they share authentication. This is called a By-Session group (session-sharing). In addition, the session may contain user defined data.
- Continuous sharing by matching group identifiers. This is called a By-ID group. In this system, the group shares a security context. The applications are usually in separate sessions.

  Important: In this system, the group does not manage user data, only authentication.

13.2 One-Time Sharing: Login Cookies

Login Cookies hold information about the most recently logged-in user. If you want to keep your users from having to log in too often, but you want your applications to remain distinct and unconnected, use Login Cookies. This corresponds to the web application option **Login Cookies**.

For Login Cookies, place each application in a separate session. Then authentication is shared only when an application is entered for the first time. Login Cookies applications do not form a group. So after login, changes in authentication in one application do not affect the other applications.

When a user logs in with a password, that authentication is saved in a cookie. If another application with Login Cookies enabled is entered (for the first time), it uses the authentication saved in the cookie. If the user jumps to a third application (for the first time) which does not have Login Cookies enabled, the user must enter a username/password.

13.3 By-Session Groups (Session-Sharing)

One way to share authentication and data between applications is to define a *by-Session group*). In this system, the authentication of the group's applications moves as a unit. If a user in an application in a group logs in as a new user, all the applications move to that user. If one application logs out, they are all logged out.

When applications share a session, they share both authentication and data via the session object.

Sharing a session has potential issues. Session events are picked up only from the original web application. If the link goes to a page that requires different session events, then these session events do not run. Also, running a page in another web application with a different security context may require a login; the login might alter the security context of running pages in the original web application. Before choosing to use By-Session groups, please read Considerations in Choosing Your Strategy below.

There are two ways to share a session:

- Session Cookie Path: All applications with exactly-matching session cookie paths are placed into the same session.
- CSPSHARE: Putting CSPSHARE=1 in the link to the application page. Use this when the source application's Session Cookie Path is different from the target's Session Cookie Path.

If By-Session sharing is required, then the best solution is to name all applications so they can be given the same Session Cookie Path. You may have to rename your applications because the Session Cookie Path must be a substring of the application name.

If this cannot be done and session sharing is required, then you have to put the CSPSHARE parameter in links that jump from one application to another. The target application page is placed in the same session as the source application's pages. The source's session is determined either from the CSPCHD parameter or the session cookie.

13.3.1 CSPSHARE

When CSP receives a request from a browser it does a series of checks to see if the sessionId it receives is a valid one. These checks include:

- Whether the User-Agent is the same as that of previous requests from this sessionId
- If cookies are being used, whether this sessionId comes from a cookie or from a CSPCHD parameter

If you pass a `CSPSHARE=1` query parameter, a sessionId cannot be passed through a URL, even if the application has been configured to only use cookies, if the "Prevent Login CSSRF attack" option has been checked. InterSystems recommends employing this behavior.

If the "Prevent Login CSSRF attack" option has not been checked and `CSPSHARE=1`, CSP performs no checks and a user can construct a link to another web application and include the current sessionId, using `CSPCHD=`*sessionId*, so that this link runs in the same session as your existing page. In addition, if `CSPSHARE=1` when you construct a link, CSP automatically inserts the `CSPCHD=`*sessionId* in to the link. If you manually insert a link with Write statements, you may need to insert the

sessionId manually. For example, if you have an application that requests an `http` page from an `https` page (or an `https` page from an `http` page), add `CSPSHARE=1` to the link as follows:

```
#(..Link()(%request.URL_"?CSPSHARE=1"))#
```

`CSPSHARE=1` forces the link construction to add CSPCHD to share the sessionId even if InterSystems IRIS detects that cookies are enabled.

See Considerations about CSPSHARE for more information.

13.4 By-ID Groups

Another way to share authentication and data between applications is to define a *by-ID group*, as follows:

1. Navigating to **System Administration** > **Security** > **Applications** > **Web Applications** on the Management Portal.
2. Give the applications a common group name in the **Group by Id** field. This name groups opened applications together.

Groups are in different sessions. The applications do not share data.

The group name is attached to an application, not a namespace. Applications with the same group name share authentication regardless of namespace.

Authentication is shared within a single browser only.

13.5 Authentication Architecture

13.5.1 Security Context & Sticky Logins

Applications are run in sessions. A session requires a security context in which to run an application. The security context contains the authentication state.

By-Sessions and By-ID Groups have a *sticky login* which remembers the security context of the last application used in the session or group. If a user in a group application logs in as a different user, the sticky login is updated. (The sticky login is not updated if the user logs in to an unauthenticated application.)

When jumping to an application in a session, the session attempts to use the sticky login appropriate for the target application. If the sticky login does not match the session's current security context and the application can accept the authentication method in the sticky login, the session's security context is switched to that in the sticky context.

A session's sticky login is lost when the session is ended. The group's sticky login is lost when all the sessions containing any of the group's applications are ended.

After the initial login, a group has an associated sticky login object which it attempts to use when entering one of the group's applications. The sticky login is not updated when an application in the group is entered as UnknownUser as this would have the effect of moving all other applications in the group to the unauthenticated security context.

If the sticky login contains a two-factor authenticated user, that two-factor authentication is used for non-two-factor applications, so long as the username authentication matches in the two applications.

13.5.2 Cascading Authentication

The CSP Server uses precedence when attempting to obtain authentication information for an application. It attempts to get new authentication information in each of the following events:

- For the first request to a new session;
- When there is an application change within the session;
- When the application is part of a By-id group and the session's current security context does not match that of the group's sticky context;
- When the request contains a username/password pair.

It attempts to get new authentication information sequentially in the following order:

1. Explicit Login: Checks to see if the user entered an authenticated username/password. If they did, the system updates the application's authentication group's context. (This sets the group's Sticky Login.)
2. Sticky Login: Get the Application's group's sticky context. If no sticky login and group-by-session, use session's current context.
3. Login Cookie: Use if one exists and is enabled for this application.
4. Unauthenticated: Use Unknown User if enabled for application.
5. Put up Login Page: If all the above fail, then request username/password from user. If called from the %CSP.Session API, then only username/password is tried. After login, update the group's sticky login unless just logged in as UnknownUser.

13.5.3 Log Out or End Session

Authentication is lost when a session is logged out or ended. You can use the following %CSP.Session methods to log out or end a session:

Recommended: CacheLogout=end

The recommended way to logout of a CSP session is to link to the application home page passing a URL that contains the string, `CacheLogout=end`. This ends the current session – releases any license acquired, deletes existing session data, and removes the security context of the session – before it attempts to run the home page.

If this web application requires authentication, there is no session and no authenticated user. In this case, IRIS does not run the home page logic but displays the login page instead. When the user submits a valid login this starts this new session and then displays the home page.

Set EndSession? =1

This kills the session. The session's sticky context is destroyed. **OnEndSession()** is called. If the session contains a By-Session group, then the group is destroyed. If the session contains a By-Id application, then that application is removed from the group which continues to exist unless this was the only application in the group. Login cookies are unaffected. By-Session groups lose their data. However, for By-Id groups, the sticky-login for the group is unaffected by a singular destruction and the other members of the group remain logged in.

In addition, for By-Session groups, the destruction *disperses* the members of the group and if the member applications are reentered, it cannot be guaranteed that they will be reintegrated into the same new session or (if they were grouped using CSPSHARE) sent to diverse sessions.

Session Logout

The session is logged out. Its sticky context is destroyed. If the session contains a by-session group, then all the applications in the group lose their authentication. If the session contains an application from a by-id group, then group loses its sticky context and all the applications in the group are logged out.

In addition, **OnLogout** is called. The login cookie is destroyed.

The session continues to exist, so data is retained for By-Session groups.

Session Logout All

It is possible to log out all session currently authenticated as a particular user.

This zaps the login cookie.

The sessions continue to exist but have not authentication.

13.6 Considerations in Choosing Your Strategy

This section contains some points to consider when you are choosing your strategy.

13.6.1 Considerations for Login Cookies

When deciding whether or not to share via Login Cookies, consider the following points:

- The login cookie is updated to a new user whenever the user logs in with a password.
- Login cookies are not generated for an unauthenticated login (as UnknownUser).
- Login cookies are not generated when logging in through API calls.
- Login cookie sessions are independent once that session has been authenticated. So logging out or timing out in one session does not affect the other sessions.
- Authentication from a Login-Cookie application cannot be shared with a password-only (non-Login-Cookie) application. For authenticated applications in a group, for consistent behavior, use Login Cookies for all or for none.

13.6.2 Considerations for Groups

This section contains some points to consider when you are creating authentication groups to share authentication.

- Use session-sharing only when you decide that data must be shared via the session object. By-ID and Login Cookies-sharing are more robust and predictable.
- When creating groups, be as consistent as possible to create uniform behavior for your targeted users. Do not place an application in both a By-ID group and a By-Session group. Using the different authentication strategies may cause unexpected behavior. By-ID takes precedence over By-Session. So if an application has both, it stays synchronized By-ID.
- Use the same authentication types for all members of the group. In particular, if some applications in the group allow Login Cookies and others do not, then entering the group via a username/password authenticates the entire group, whereas entering it via a login cookie authenticates only some of the applications. This can cause confusion among your users about why sometime a login is required and other times not.
- The CSP server considers every application to be in an Authentication Group. A lone application in a session forms a single-entity By-Session authentication group.)

- Try not to put unauthenticated-only applications in By-ID groups.
- By-Session groups are fragile; using By-ID is a more robust approach. Since all the information about a group, including its shared data, is contained in a single session, the group can easily be lost. This is because a session can time out, that is, after a specific amount of time the session is automatically destroyed. If the user steps away from his computer or uses an application which is not in the By-Session group, the session may timeout. If one of the applications in the group marks ENDSESSION=1, the group is dispersed.
- If the browser has open tabs containing pages from the dispersed applications, clicking on them may require multiple logins, especially if they were originally grouped using CSPSHARE=1. In any case, the data from the original session is permanently gone.

  When a group loses its authentication, refreshing or going to an open page from a group application requires that the user re-login.
- Ending a session containing a By-Session application requires that the user re-login when refreshing any page of any application in that by-session group. Killing a session containing a By-ID application does not require any logins unless that session's application was the only member of the group.
- Logging out a session logs out all members of the session's group, even if they are in different sessions. Refreshing any of the group's pages requires a new login. However, for By-ID groups, one login logs in the entire group. For By-Session groups, one login logs in the entire group as long the Web Gateway is able to direct the dispersed applications back to a newly constructed session object.
- Logging out does not destroy the session, so any session data continues to exist.
- One cannot have same application logged in to two different users in different tabs of the same browser.
- Authentication is shared within a single browser only. This runtime identifier is stored in the %Session object.
- Grouping allows you to share authentication with users that are in the same group (By-ID) or the same session (By-Session). If you want to share authentication from applications that are outside your specified group, use Login Cookies. If you want to send authentication to applications outside your specified group, use CSPSHARE=1. (See Considerations about CSPSHARE.)

13.6.3 Considerations about CSPSHARE

Use CSPSHARE as a last resort.

By-Session application links do not need CSPSHARE=1 in the following cases:

- If the source and target applications have the same group ID.
- If the target page is in the same application as the source page.
- If the target page application's Session Cookie Path matches the source application's Session Cookie Path.

13.6.4 Sharing Data

By-Session groups can share data via the session object.

By-ID groups must manage their own data. If the data is stored, for example, in a global, the data could be keyed using the current user, $Username, or by the group's runtime ID. The CSP Server assigns each browser a browser-id cookie. When a By-Id group is created it is assigned a key which is the browser ID concatenated with the group ID. This creates a unique key, %CSP.Session.BrowserId, which can be used as a key under which to store data.

14

Enabling Logging

This page describes how to enable logging that records CSP activity, which is useful for troubleshooting CSP-based web applications.

14.1 Enabling and Disabling Logging

Enable logging by entering the following command in the Terminal:

ObjectScript

```
Set ^%ISCLOG = 2
```

You can view logging information in the **^ISCLOG** global.

You can turn logging off with either of the following commands:

ObjectScript

```
Set ^%ISCLOG = 0
Kill ^%ISCLOG
```

14.2 Log Levels

For reference, the log levels are as follows:

- 0 — InterSystems IRIS® data platform performs no logging.
- 1 — InterSystems IRIS logs only exceptional events (such as error messages).
- 2 — InterSystems IRIS logs detailed information, such as `method ABC invoked with parameters X,Y,Z and returned 1234`.
- 3 — InterSystems IRIS logs raw information such as data received from an HTTP request.
- 5 — InterSystems IRIS logs OAuth 2.0 information.

14.3 ISCLog Details

In ISCLOG, some entries match Event Log header fields as follows:

ISCLOG	Event Log
Job	Cache-PID
SessionId	Session-ID
Tag	Request-ID

Fields and definitions in ISCLOG are shown in the table below.

Table 14–1: ISCLOG Fields

Field	Definition
%category	`CSPServer`: Logged from `cspServer`, `cspServer2`, `%request`, `%response`.
	`CSPSession` Logged from `%session` and parts of `cspServer` and `cspServer2` which handle a session. This allows watching the lifecycle of a session.
	`CSPLicense`Logged from parts of `cspServer` and `cspServer2` which handle a licensing.
	`Gateway Request`Logged from the `GatewayMgr`, `GatewayRegistry`, the Gateway request handler and parts of `cspServer2` which handle gateway requests.
%level	1= Exceptions and errors.
	2=`CSPSession` information. `CSPLicense` information. Information from `cspServer`: the part of the request handling after the `%response`, `%session`, and `%request` have been setup. This includes authentication, license handling, redirection, and calling the CSPpage.
	3=Information from `cspServer2`: the part of handling the request which sets up the `%response`, `%session`, `%request`, and hand-shaking/data transfer with the Web Gateway.
%job	The value of `$job` when the ISCLOG request was made. Matches the Cache-PID field from the Event Log header.
%sessionid	Entered when available. The value of `sessionid` at the time the ISCLOG request was made. Matches the Session-ID field from the Event Log header.

%tag	For the CSP Server, the tag contains the Request id from the gateway (when available). This matches the `Request-ID` field from the Event Log header. Other loggers may set this value to any value.
	Available for use by creators of ISCLOG entries. Stores ID of the request sent to it by the Web Gateway. It can be used as a filter for generation of ISCLOG entries.
	`Set ^%ISCLOG("Tag","mytagvalue1")=1`
	`Set ^%ISCLOG("Tag","mytagvalue2")=1`
	Only ISCLOG requests with no tag or with tags of "mytagvalue1" or "mytagvalue2" will be recorded.

Field	Definition
%routine	The name of the routine currently being executed.
%message	See Message Format below.

14.4 Message Format

Messages start with the name of the tag label or method currently being executed. This name is enclosed in square brackets. `[MyMethod] rest of messages`.

Messages in the `CSPSession` category also have `CSPSession-Id=sessid` after the method name. This is needed as session events can be logged before the session is created or after it was destroyed, meaning the SessionId field is empty in the ISCLOG entry.

```
[MyMethod] CSPSession-Id: 12ty34ui22
```

Messages in the `GatewayRegistry` category also have `CSPID=cspid`(when available) after the method name. This allows the tracking of an individual gateway request from the API call through the Gateway Request Handler.

```
 [MyMethod]CSPID:334r43345 rest of message
```

A

Reserved URL Parameters

The following URL parameters are reserved, within the context of a CSP-based web application:

IrisUserName

From the login page, contains the username to log in

IrisPassword

From the login page, contains the password of the user designated by `IrisUserName`

IrisOldPassword

If passed in with `IrisUserName` and `IrisPassword`, it contains the current password for the user. The security routines changes the user's password to a new value, the one from `IrisPassword`, such as, `IrisOldPassword=fredsAboutToBeChangedPwd`. After the password is changed, the user is logged in using the new password.

IrisLogout

`IrisLogout` with no value or any value other than `cookie` causes the session for this request to be logged out (but not destroyed.) Logging out destroys the current login cookie and removes any two-factor security tokens being held in limbo for this session.

`IrisLogout=cookie` destroys the current login cookie.

IrisSecurityToken

`IrisSecurityToken` contains the value of a submitted security token from the Login Security Token page, such as `IrisSecurityToken=12345678`.

IrisSecuritySubmit

The presence of this name indicates that the user is submitting a security token whose value is associated with `IrisSecurityToken`.

IrisSecurityCancel

The presence of this name indicates that the user has cancelled out of the Login Security Token page.

IrisLoginPage

Login pages, including custom login pages, contain two sub-pages: one for login and one for returning the security token value. The page checks the value of `IrisLoginPage` to determine which subpage to display. `IrisLoginPage=1` indicates the Login subpage should be displayed.

IrisNoRedirect

A page `P` is requested, but it is unauthenticated, so the Login page is displayed. After the user submits the information from the login page, usually the page request for `P` is redirected back to the browser. (This stops the browser from asking the user to press the `<Resend>` button before its shows `P`.) This behavior can be short-circuited by passing `IrisNoRedirect=1`

B

Special HTML Directives

In the **OnPage()** callback in a CSP page class, you can use the special directive &html<> to write HTML. This directive can be a simpler way to construct HTML, although the result does not lend itself to unit testing (because you cannot examine the result without using a browser).

B.1 &html<> Basics

The special directive &html<> can contain any valid HTML, including multiple lines of HTML. For example:

Class Definition

```
Class GCSP.HTML Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   &html<<!DOCTYPE html>
   <html lang="en" dir="ltr">
   <body>
   <h1>Basic Page</h1>
   <div>No double quotes needed "here"</div>
   </body>
   </html>>

   Quit $$$OK
}

}
```

B.2 Expressions within &html<>

Sometimes it is convenient to include expressions *within* the HTML contained in the directive &html<>. There are two possibilities:

- Expressions evaluated at compile time. For this, you can use the following syntax:

  ```
  ##(expression)##
  ```

  This kind is known as a *CSP compile-time expression*.

- Expressions evaluated at run time. For this, you can use the following syntax:

  ```
  #(expression)#
  ```

Where *expression* is any ObjectScript expression. This kind is known as a *CSP run-time expression*.

In either case, simply include the syntax at the position where the expression is needed (not performing any sort of concatenation). Here is an example:

```
Class GCSP.HTML1 Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   &html<<!DOCTYPE html>
   <html lang="en" dir="ltr">
   <body>
   <h1>Basic Page</h1>
   <div>This class was compiled at ##($zdatetime($h,3))##</div>
   <div>This class was viewed at #($zdatetime($h,3))#</div>
   </body>
   </html>>

   Quit $$$OK
}

}
```

C

CSP Error Codes

This page describes causes of errors that can occur within a CSP-based web application, as well as approaches to solving them. Some of these errors apply only in tag-based development but are included here for simplicity.

Table III–1: CSP Error Codes, Error Messages, and When Reported

Error Code	Error Message	When Reported
5902	Rule '%1' does not exist	Reported when calling `%apiCSP` to add attributes to a rule if you specify a rule name that does not exist.
5903	Rule name is required	Reported if you attempt to add or delete a rule but do not provide a name for the rule.
5904	Attribute '%2' is required for tag '<%1>' on line number %3	Reported if you did not supply a required attribute for a tag in the CSP page. The page cannot compile without this required attribute.
5905	The value of attribute %1, '%2', is invalid, on line number %3	Reported if the value of an attribute in a CSP page is not a valid choice. For example if you define `<script language="Cache" runat="XXXXX">`, the `runat` value is not a valid choice. The CSP compiler cannot compile this page and reports this error.
5906	Session ID is missing	Reported if you attempt to create an instance of %CSP.Session without supplying a session ID in the **%New()** method. For example, `Set session=##class(%CSP.Session).%New()` reports this error but `Set session=##class(%CSP.Session).%New(1234)` does not as it passes the session ID `1234`.
5907	Session ID '%1' does not exist	Reported if you attempt to load an existing **%CSP.Session** but pass the %OpenId() a session ID that is not stored in InterSystems IRIS® data platform.
5908	Failed to create class '%1': %2	Reported by the CSP compiler if it cannot create the class corresponding to the CSP page.
5909	There is no closing tag for the tag <%1> on line number %2	Reported if the CSP compiler detects that you opened a tag but never closed it (if the tag specifies that it needs a closing tag in the rule definition).

Error Code	Error Message	When Reported
5911	Character Set '%1' not installed, unable to perform character set translation	Reported if the character set specified in the CSP page to output this page is not installed in InterSystems IRIS. This could be the character set specified in the %response.CharSet property in the **OnPreHTTP()** method. See the charset property of the class %CSP.Page. Check that you intend to use the character set reported in the error and if so, check that this is installed in InterSystems IRIS. or by setting the %response.CharSet property in the **OnPreHTTP()** method.
5912	CSP Page '%1' does not exist	Reported if you request a CSP page that does not exist. You may have mistyped a URL or a link on another CSP page may be incorrect. Check if the page exists on the server and, if not, then look for where the link came from. If the page should exist, make sure the web application settings are correctly set to point to the right directory and check that the CSP file exists on the disk. This error only occurs if the `autocompile` option is on and the CSP engine tries to compile this page and cannot find the file.
5914	Web Application '%1' does not exist	Reported when the application part of the URL cannot be found in the web application list. For example, you try to load the page /cspx/samples/menu.csp with a type of `cspx` rather than `csp`, then InterSystems IRIS cannot find the web application. Check the list of applications by navigating to **System Administration** > **Security** > **Applications** > **Web Applications** in the Management Portal and check the command for mistakes.
5915	Cannot allocate a license	Reported if the license limit has been reached so this new request for a CSP session cannot be granted. You may be able to reduce the default timeout on CSP sessions specified in the web application configuration or you need to look at buying more licenses.
5916	Illegal CSP Request	Reported when you try to reach a private page by entering the URL instead of being redirected from another CSP page which includes the encrypted token to allow access to this page, or by using an invalid encrypted token to allow access to this private page.
5917	HTTP method '%1' not supported by CSP	Reported when you attempt to use an unsupported HTTP method. HTTP methods supported are GET, POST, HEAD. We do not support other HTTP methods in the CSP server at present. It can also be caused by an incompatible version of the Web Gateway talking to the CSP server.

Error Code	Error Message	When Reported
5918	You are logged out, and can no longer perform that action	Reported if the CSP request contains encrypted data, but the session is a brand new session, so there is no way that the decryption key can match the encrypted data. Typically this is because the session has timed out. Then the user subsequently does something in the browser to cause another request. You can increase the session timeout value or use the error mechanism to redirect the user to an initial page so they can start their action again.
5919	The action you are requesting is not valid	Reported typically when passing an encrypted string to InterSystems IRIS from the CSP page where the decryption key does not match the key used to encrypt this data. This can be caused by the user tampering with the URL manually or by anything that could change the value of the encrypted string between it being generated in InterSystems IRIS and returned back to InterSystems IRIS in the next HTTP message.
5920	Must run this CSP page from namespace '%1'	Each web application is tied to a specific namespace in InterSystems IRIS. This error is reported if you attempt to do something such as compiling a page from /csp/samples/loop.csp in the USER namespace when the /csp/samples application is tied to the SAMPLES namespace.
5921	The web application '%1' must specify a namespace to run in	Reported if the configuration of the web application is missing the namespace. This generally indicates that the CPF file has been badly edited by hand as the Management Portal does not allow a web application to be created without a namespace.
5922	Timed out waiting for response	Reported by the %Net.HttpRequest object when it times out waiting for a response from the HTTP server it is talking to.
5923	Redirected %1 times, appears to be a redirection loop	Reported If more than 4 redirects are detected in one page. The compiler assumes that there is a loop. If a CSP page uses the **ServerSideRedirect** to jump to another page there is a possibility that page A.csp could redirect to B.csp which redirects to A.csp creating a loop.
5924	An error occurred and the specified error page could not be displayed - please inform the web master	When an error in a CSP page occurs at runtime, the CSP engine redirects to a user-specified error page that can handle the error in any manner it wishes. If, however, this user-specified error page does not exist or there is an error in generating this error page, then the CSP engine logs the fact that something has gone wrong using BACK^%ETN and reports this error message. As this error may appear on a production system if there is a bug in the user— written error page, the message is deliberately vague. To resolve this error, first check that the error page specified in the web application exists and then look at possible bugs in this error page.

Error Code	Error Message	When Reported
5925	<SCRIPT LANGUAGE=Cache> tag is missing either RUNAT or METHOD attribute, on line number %1	Reported if the `<script language="Cache">` tag is missing the required attribute *runat* (to tell the CSP compiler when this code should run), or the*method* attribute to create a new method.
5926	Unable to redirect as HTTP headers have already been written and flushed	Reported if you try to use a server side redirect after data has been written to the browser .If you attempt to use the *%response.ServerSideRedirect* feature to redirect to another page, this must be done before any data has been written back to the browser. Typically this means you must do this in the **OnPreHTTP()** method of the page.
5927	Unable to load page '%1' because its class name conflicts with the class '%2' that is already loaded	Reported if you have two CSP files with identical names in different applications in the same namespace: For example, if you have two CSP applications, `/test` and `/anothertest`, both in the USER namespace. which are in different directories on the InterSystems IRIS server, each of which has a file test.csp. If you have `autocompile` turned on and you enter the URL /test/test.csp the CSP compiler compiles this page into the class csp.test. If you enter the URL /anothertest/test.csp, it tries to load this page to create the class csp.test, finds it already exists for a different application and reports this error. If it did not do this, you would see very poor performance as each request would recompile the entire page. Either avoid using identical file names in the same namespace or change the package defined in the web application, which defaults to `csp`. For example, change`/anothertest` to use package name package. Then when it compiles test.csp, it creates the class name package.test which does not conflict with the other application that uses csp.test.
5931	Can only call this method/set this value in OnPreHTTP() before page has started to be displayed	Reported if you call a function that needs to be called in the **OnPreHTTP()** method of the page so that it can modify some parameters before any data is output to the browser. Move this call to the **OnPreHTTP()** method to resolve this.
5932	Action not valid with this version of the Web Gateway on the web server	Reported if the version of the Web Gateway you are using does not support this action. Either do not use this feature or upgrade the version of the Web Gateway to a later version.
5933	The CSP server had an internal error: %1	Reported if an unexpected error condition has occurred inside the CSP engine. Please report this to InterSystems support.
5954	Failed to lock CSP page.	When a CSP page is autocompiled it is first locked to make sure that two jobs do not both attempt to compile the same page at the same time. If the lock is not released by the other job in 60 seconds, it assumes the compile failed for some reason and reports this error message. Try recompiling this page from Studio to see if any errors are reported.
5955	CSPAppList query: invalid data in Fetch().	Reported if the query to determine the list of CSP applications is invalid. This error should never be seen on a working system.

Error Code	Error Message	When Reported
5956	Directory '%1' for Web Application '%2' does not exist	Reported if the directory pointed to by the web application does not exist in the file system.
5961	Unable to convert character set '%1'.	Reported when a request from a browser comes in. The information sent by the browser is converted into the current InterSystems IRIS default locale and there is an error. To debug the conversion, isolate the information being sent by the browser and convert it from that character set manually in a test program.
5962	Unable to allocate new session.	Reported when calling **%session.ForceNewSession()** if there are no new slots in this session Id.
5963	Invalid SysLog level: %1.	Reported when setting the internal log level if the level is outside the allowed range.