

Caché Server Pages QuickStart

Version 2.0

Последнее изменение: 13 октября 2003
Copyright © InterSystems Corp, 2000-2003

О курсе Caché Server Pages QuickStart

Курс CSP QuickStart предназначен для тех, кто хочет в кратчайшие сроки самостоятельно начать использовать Web-технологии Caché. Среднее время прохождения курса – полдня.

Для прохождения курса необходимо знание основ HTML и Caché.

Для того чтобы научиться работать с классами Caché рекомендуется предварительно ознакомиться с документом Object Quick Start.

Оглавление

О курсе Caché Server Pages QuickStart	2
Установка и настройка CSP	3
Hello World!	4
Мастер Форм Caché	6
Теги CSP-приложения	6
Генерация XML-документов с помощью CSP	8
Обработка запросов пользователей, %request	9
Работа с сессией, %session	10
Гипер-события	11
Привязка объектов к форме. Тег CSP:OBJECT, CSPBIND	12
Шифрование и безопасность	14
Include	14
Создание собственных тегов	15
Что дальше?	16
Приложение 1. Советы по отладке CSP-приложений	16
Приложение 2. Интеграция с Macromedia DreamWeaver.	17
Приложение 3. Дополнительные источники информации по CSP	17

Установка и настройка CSP

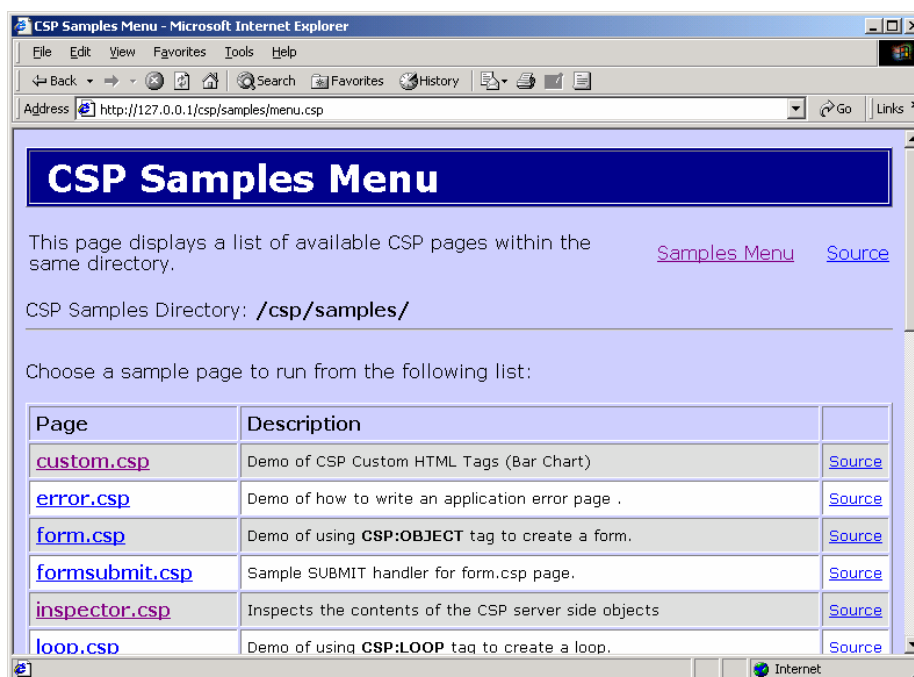
Этот раздел рассчитан на случай, когда Вы устанавливаете Caché на Ваш локальный компьютер с операционной системой Windows и установленным Web-сервером Internet Information Server или Personal Web Server. Также возможен вариант, при котором используется встроенный в Caché Web-сервер. В других случаях, обращайтесь к документации по CSP.

При установке Caché автоматически определяет, установлен ли на Вашем компьютере Web-сервер и устанавливает все необходимые компоненты. Поэтому рекомендуем перед инсталляцией Caché убедиться, что на Вашем компьютере уже установлен и работает Web-сервер. Для этого проверьте следующий адрес: <http://127.0.0.1> . Это адрес Вашей локальной машины. Если Web-сервер на Вашей машине установлен, Вы должны увидеть домашнюю страницу сервера.

После инсталляции Caché проверьте, работает ли CSP. Для этого введите в строке браузера следующий адрес:

<http://127.0.0.1/csp/samples/menu.csp>

Если CSP работает корректно, Вы должны получить следующую картинку:



Это меню примеров CSP. В будущем Вы можете использовать эти примеры для самообразования.

Если Вы не смогли получить данную картинку, то CSP не установлен корректно или на вашем компьютере в момент установки Caché не был установлен Web-сервер. Вы можете самостоятельно установить CSP позднее, руководствуясь документацией по CSP.

Для тестирования и изучения CSP Вы также можете использовать встроенный в Caché Web сервер, использующий для своей работы порт 1972. Вы можете получить доступ к меню примеров CSP по адресу:

<http://127.0.0.1:1972/csp/samples/menu.csp>

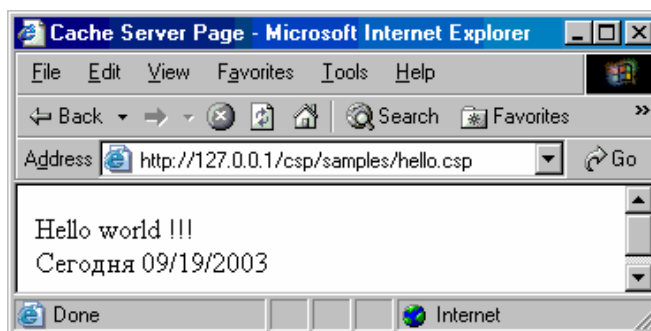
Как видите, это адрес ничем не отличается от адреса, введенного ранее, кроме явного указания на порт http-сервера - 1972. В дальнейшем все примеры данного курса будут вызываться, указывая порт 1972 в соответствующих URL. Если Вы настроили какой-либо промышленный Web-сервер для работы с CSP, то можно работать с ним, не указывая порт 1972.

Функциональность встроенного в Caché Web сервера достаточна для изучения CSP и даже для обеспечения процесса разработки (с рядом ограничений), однако мы настоятельно не рекомендуем использовать этот сервер для эксплуатации Ваших приложений.

Hello World!

Создайте в блокноте Windows текстовый файл, приведенный ниже, и сохраните его в каталоге C:\CacheSys\CSP\Samples (Если Вы установили Cache не в директорию CacheSys, то сохраните файл в соответствующую директорию).

Теперь в браузере обратитесь к странице: <http://127.0.0.1:1972/csp/samples/hello.csp> Вы увидите следующее:



Правой кнопкой мыши щелкните в браузере и выберите пункт меню View Source. Вы увидите исходный код страницы, пришедшей из сервера в браузер. Как видите, в браузер попадает текст HTML, в котором вместо конструкции #{\$zd(\$h)}# стоит значение текущей даты. То есть на сервере Cache происходит обработка исходного кода CSP файла.

Чтобы выяснить, как конкретно работает внутренний механизм CSP, откройте в Cache Studio в Namespace «SAMPLES» класс csp.hello:

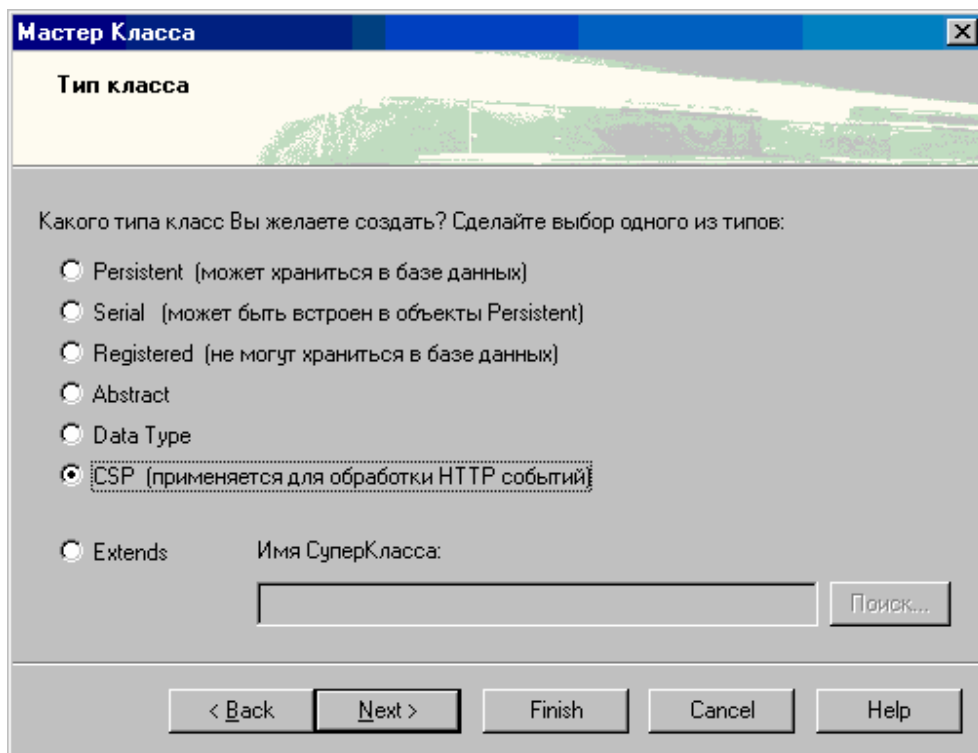
Посмотрите на код метода OnPageBODY(). Как видите, на базе файла hello.csp был создан класс csp.hello, в методах которого и заключен код, генерирующий страницу, попадающую в браузер.

Как видите, каждая страница CSP-приложения представлена классом в Caché. Класс – это не просто способ представления кода. Это отражение полноценной объектной модели CSP-приложения. Вы можете использовать все преимущества объектного подхода к разработке при работе с CSP. Например, наследование и полиморфизм.

Как Вы уже успели заметить, каждая CSP-страница преобразуется в класс Caché. Вы также можете создавать классы непосредственно в Caché Studio, не создавая файл .csp.

Для дальнейшей работы в Namespace «SAMPLES» создайте проект (логическое объединение классов, программ и CSP-страниц) CSPQuickStart и добавьте в него класс csp.hello и страницу hello.csp.

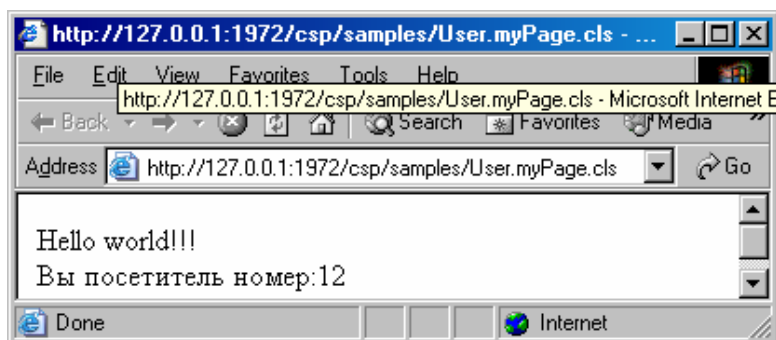
Создайте в Caché Studio новый класс User.myPage, типа CSP Page:



Модифицируйте метод ..OnPage нового класса:

```
ClassMethod OnPage() As %Status
{
    &html<<html>
    <head>
    </head>
    <body>>
    Write "Hello world!!! <br>",!
    Write "Вы посетитель номер:",$i(^a),!
    &html<</body>
    </html>>
    Quit $$$OK
}
```

Сохраните и скомпилируйте класс. Откройте соответствующую CSP-страницу в браузере с помощью Caché Studio (Вид → Web-страница) или набрав в браузере адрес: <http://127.0.0.1:1972/csp/samples/User.myPage.cls>



Обратите внимание, что при обращении непосредственно к классу расширение соответствующей страницы меняется на .cls

Мастер Форм Caché

Вы можете создавать CSP-страницы с использованием произвольного редактора HTML-страниц или в Caché Studio. Для создания простой формы для работы с классом можно использовать Мастер Форм (Caché Form Wizard).

Для этого нужно запустить Caché Form Wizard и выполнить следующие действия:

1. Добавьте в проект CSPQuickStart класс Sample.Person.
2. Создайте новую CSP-страницу в проекте.
3. Удалите текст внутри тега <BODY>
4. Вызовите Мастер Форм (Caché Studio → Вставить → Мастер Форм)
5. Выберите свойства класса Sample.Person, которые должны быть представлены на форме
6. Для каждого атрибута Вы можете задать свойство только «для чтения» (read-only) и изменить заголовок (В данном примере измените Заголовок свойства Name на «Имя»).
7. Закончите генерацию формы.
8. Сохраните страницу как Wizard.csp в csp/samples

Откройте соответствующую CSP-страницу в браузере с помощью Caché Studio (Вид → Web-страница) или набрав в браузере адрес: <http://127.0.0.1:1972/csp/samples/wizard.csp>.

Вы можете добавлять и изменять записи в классе Sample.Person, а также производить поиск.

Как видите, Вы получили обычный html-файл, в который встроены специфические расширения – CSP-теги. Например, CSP:OBJECT, CSP:SEARCH и нестандартные параметры обычных тегов, например, CSPBIND для тегов HTML-формы.

Разработка CSP-приложения в большинстве случаев представляет собой создание обычных HTML страниц, в которые внедрены специфические CSP-теги. В следующих главах Вы сами будете создавать CSP-страницы.

Теги CSP-приложения

Теги CSP обеспечивают значительное расширение функциональных возможностей при создании Web-приложений. В Таблице 1 представлены основные CSP-теги.

Таблица 1

Дополнительные функциональные возможности, предоставляемые CSP-тегами

Вставка данных:	
#(<Expression>)#	<Expression> - выражение Caché ObjectScript, возвращающее значение
##(<Expression>)##	То же, но во время компиляции

Управление	
<CSP:IF CONDITION='a=1'> Unauthorized!!! </CSP:IF>	Условие
<CSP:WHILE ...> <CSP:LOOP ...>	Циклы
Использование Caché Script	
<SCRIPT Language="Cache" RUNAT="Server/Compiler"> </SCRIPT>	Блок Caché ObjectScript кода. runat=«server»: блок обрабатывается на этапе выполнения или генерации HTML-страницы. runat=«compiler»: блок обрабатывается при компиляции. Логика может быть описана независимо и может влиять на вид HTML-страницы. (Выражения после "Write" отображаются на стандартном устройстве вывода – по умолчанию - браузер)
<SCRIPT Method=methodName Arguments=spec [ReturnType=dataType]>Inner Text</SCRIPT>	Метод CSP-класса
Запросы к БД	
<CSP:QUERY ...> <CSP:SEARCH ...> <SCRIPT LANGUAGE="SQL" ...>	Запрос класса Поисковая форма SQL-запрос
Привязка объекта к форме	
<CSP:OBJECT ...> <FORM CspBind="obj" ...> <INPUT CspBind="obj.Name" ...>	Открытие объекта Привязка объекта к форме Привязка свойства к полю
Управление параметрами класса	
<CSP:CLASS [Encoded=encodedType] [Private=accessType] [Super=classList] ...>	Определение родительских классов, режима шифрования и пр.

Список всех активных тегов можно получить, набрав <http://127.0.0.1:1972/csp/samples/rulemgr.csp>

Для иллюстрации возможностей использования тегов CSP Вы можете создать две CSP-страницы: Person.csp и PersonSearch.csp.

Создайте в Caché Studio новую CSP-страницу Person.csp:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <script language="Cache" runat="Server">
    set obj=##class(Sample.Person).%OpenId(1,0)
```

```

        write obj.Name, "<br>"
        write obj.Age, "<br>"
        kill obj
    </script>
</BODY>
</HTML>

```

CSP-страница должна выдавать информацию об экземпляре объекта Person с Id=1.

Таким же образом Вы можете создать CSP-страницу PersonSearch.csp, которая выводит на экран браузера все экземпляры класса Person.

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<csp:query name="query" classname="Sample.Person" queryname="ByName" P1="">
<csp:while Condition="query.Next()">
    <li> #(query.Get("Name"))#
</csp:while>
</BODY>
</HTML>

```

Откройте PersonSearch.csp в браузере (<http://127.0.0.1:1972/csp/samples/personsearch.csp>).

Генерация XML-документов с помощью CSP

CSP-страница может возвращать в браузер не только HTML, но и XML.

Создадим CSP-страницу, которая возвращает XML-документ с информацией обо всех экземплярах класса Sample.Person. Так как класс Sample.Person наследник %XML.Adaptor, можно использовать метод XMLExport() для экспорта каждого экземпляра класса в XML.

Создайте в Caché Studio новый класс csp.XMLServer типа CSP Page. Выберите тип контента XML. Модифицируйте метод OnPage нового класса:

```

ClassMethod OnPage() As %Status
{
    Write "<?xml version="1.0" ?>", !
    write "<People>", !
    set rs=##class(%ResultSet).%New("Sample.Person:Extent")
    do rs.Execute()
    for {if ('rs.Next()) quit
        set id=rs.GetData(1)
        set ref=##class(Sample.Person).%OpenId(id,0)
        do ref.XMLExport()
        kill ref
    }
    kill rs
    write "</People>", !
    Quit $$$OK
}

```

Сохраните и скомпилируйте класс. Посмотрите соответствующую CSP-страницу в браузере <http://127.0.0.1:1972/csp/samples/csp.XMLServer.cls>. Более подробную информацию по работе с XML можно найти в документации (Документация → Developing Applications with Caché → Using XML with Caché).

Обработка запросов пользователей, %request

В предыдущих главах Вы научились строить приложения, которые позволяют создавать Web-страницы исходя из содержимого базы данных. Все, что нам осталось сделать для создания полноценных интерактивных Web-приложений – это научиться обрабатывать запросы пользователей.

Все параметры, которые пользователь, так или иначе, передает на страницу, попадают в объект %request, класса %CSP.Request. Параметры могут передаваться как через URL (например – с использованием гиперссылок), так и через поля HTML форм.

Например, URL <http://127.0.0.1:1972/csp/samples/Person.csp?oid=2> содержит входной параметр oid, значение которого равно 2.

Выражение %request.Data("oid",1) возвращает 2. Если Вы не знаете, был ли параметр oid определен, то можно использовать выражение \$Get(%request.Data("oid",1),1), которое, если бы параметр oid не был определен, возвратило бы значение по умолчанию – 1.

Для того чтобы разобраться с возможностями объекта %request, усовершенствуем наши CSP-страницы PersonSearch.csp и Person.csp. (изменения по сравнению с предыдущими версиями выделены жирным шрифтом).

Новый текст Person.csp:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <script language="Cache" runat="Server">
    set obj=##class(Sample.Person).%OpenId($Get(%request.Data("oid",1),1),0)
    write obj.Name,"<br>"
    write obj.Age,"<br>"
    kill obj
  </script>
</BODY>
</HTML>
```

Попробуйте ввести новые значения параметра oid. Например: <http://127.0.0.1:1972/csp/samples/Person.csp?oid=2>

Новый текст PersonSearch.csp:

```
<HTML>
<HEAD>
</HEAD>
<BODY bgcolor="#FFFFFF">
<form action="PersonSearch.csp">
  <input name="SearchFor">
  <input type="Submit">
</form>
<csp:query name="query" classname="Sample.Person" queryname="ByName"
P1='#($Get(%request.Data("SearchFor",1),""))#!'>
<csp:while Condition="query.Next(">
<li><a href='person.csp?oid=#(query.Get("ID"))#!'> #(query.Get("Name"))#</a>
</csp:while>
</BODY>
</HTML>
```

В результате получено небольшое CSP-приложение. На CSP-странице PersonSearch.csp пользователь может искать экземпляры класса Person, удовлетворяющие критерию запроса, и, переходя по ссылке на страницу Person.csp, выводить подробную информацию об интересующем его экземпляре.

Объект %request содержит все данные, которые, так или иначе, передаются Вашей странице. В том числе поля заполненной на предыдущей странице формы, параметры из URL, переменные CGI, значения COOKIE и пр. Объект %request класса %CSP.Request обладает рядом полезных свойств и методов. Более подробную

информацию о классе %CSP.Request можно найти в документации (Документация Caché → Using Caché Server Pages (CSP) → HTTP Requests → The %CSP.Request Object).

Работа с сессией, %session

Приложение CSP, как и любое другое Web-приложение, использует протокол HTTP. Одна из основных трудностей при разработке Web-приложений состоит в том, что при работе по протоколу HTTP соединение между браузером и сервером прекращается сразу после окончания вывода очередной страницы. Таким образом, мы не можем определить, какие действия выполнял пользователь на предыдущих страницах нашего Web-приложения, то есть сохранять контекст приложения.

Полезной возможностью CSP является способность Caché сохранять контекст процесса от одного запроса к другому с использованием объекта %session.

Объект %session содержит ряд свойств, методов и параметров, которые помогают разработчику управлять сессией. Вы можете ознакомиться с документацией объекта %session класса [%CSP.Session](http://127.0.0.1:1972/apps/documatic?CLASSNAME=%25CSP.Session) (<http://127.0.0.1:1972/apps/documatic?CLASSNAME=%25CSP.Session>).

Мы же рассмотрим основные возможности объекта %session: сохранение данных и поддержка состояния.

Приложения могут сохранять данные как пары имя/значение в объекте %session. Например: команда `Set %session.Data("id")=1` присваивает переменной `id` значение 1. Позже, в ходе обработки страницы, можно получить значение `id`: `%session.Data("id")`.

Для иллюстрации возможностей CSP по поддержке сессии выполним следующий пример.

Создадим CSP-страницу `Login.csp`:

```
<HTML>
<form name="form1" action="PersonSearch.csp" method="post">
  Your name:
  <input type="text" name="Name">
  <input type="submit" value="Submit">
</form>
</html>
```

В реальных CSP-приложениях нужно использовать метод класса [%CSP.Session](http://127.0.0.1:1972/apps/documatic?CLASSNAME=%25CSP.Session) `%CSP.Session.Login(UserId,Password,[0 or 1])` для регистрации входа пользователя в систему.

Усовершенствуем `PersonSearch.csp`, добавив следующий код:

```
<HTML>
<HEAD></HEAD>
<script language="cache" runat="server">
  if '$Data(%session.Data("UserName",1)) {
    set %session.Data("UserName",1)=$Get(%request.Data("Name",1))
  }
</script>
<body bgcolor="#FFFFFF">
Hello #($Get(%session.Data("UserName",1)))# !
...
</html>
```

Откройте `login.csp` в браузере (<http://127.0.0.1:1972/csp/samples/login.csp>).

Теперь после регистрации пользователь попадает со страницы `Login.csp` на страницу `PersonSearch.csp`, и на ней выводится имя этого пользователя. Заметьте, что теперь Вы можете в любой момент вернуться на страницу `PersonSearch.csp` и увидеть там введенное ранее имя. В отличие от объекта %request, который создается заново при каждом обращении к CSP странице, объект %session существует в течение всего сеанса работы пользователя с приложением.

Если Вы присвоите свойству `%session.Preserve` значение 1 (`set %session.Preserve=1`), то от одного запроса к другому будет передаваться не только информация `%session object`, но и все состояние сессии (локальные переменные, содержание открытых объектов и т. д.)

Гипер-события

Одним из основных недостатков традиционных средств разработки Web-приложений является необходимость перезагрузки всей страницы, если нужно изменить содержимое ее части. JavaScript, Объектная модель документа и Динамический HTML решают эту проблему, но только отчасти. Они позволяют изменять содержимое страницы динамически, но не избавляют от необходимости полной перезагрузки страницы, в случае, когда необходимо отобразить в браузере данные из СУБД.

Технология Гипер-событий (Hyper-Events) позволяет изменять содержимое страницы без ее перезагрузки, причем эти изменения будут оперировать данными, динамически получаемыми с сервера базы данных.

Реализуем простейшую программу, предназначенную для пересчета курсов валют. Для пересчета нам необходимо знать дату, для которой делается расчет и сумму валюты в долларах.

В программе будет использован CSP-тег `<script language="Cache" method="NAME"> arguments="test:%String"> CODE </script>`, который предназначен для написания методов на Caché ObjectScript. Метод будет выполняться на сервере.

Создайте массив курса валют за месяц. Для этого в Caché терминале наберите следующую команду: `For i=1:1:31 set ^CurrencyRate(i)=30+$random(5)`. `^CurrencyRate` – это многомерный массив Caché, глобал. Более подробную информацию о глобалах Вы можете найти в документации (Документация → Developing Applications with Caché → Using Caché Multi-Dimensional Storage).

Создадим CSP-страницу `converter.csp`:

```
<HTML>
<HEAD>
<TITLE>Конвертор валют</TITLE>
</HEAD>
<BODY>
  <h3>Конвертер валют</h3>
  <form name="Converter">
    Число месяца: <input type="TEXT" name="Day"><br>
    Доллары: <input type="TEXT" name="USD"><br>
    Рубли: <input type="TEXT" name="RUR"><br>
    <input name="Go" type="Button" value="Вычислить"
      onClick="#server(..Convert(self.document.Converter.Day.value,
        self.document.Converter.USD.value))#">
  </form>
</BODY>
</HTML>
<script language="Cache" Method="Convert"
Arguments="day:%Integer,usd:%Integer">
  new rub
  set rub=usd*$get(^CurrencyRate(day),30)
  &javascript<self.document.Converter.RUR.value=#(rub)#;>
</script>
```

Как видно в примере, на стороне сервера создается метод `Convert()`, который и вычисляет значение суммы в рублях. Вызов метода происходит по событию `onClick` кнопки “Вычислить”. При этом метод не возвращает значение, а явным образом формирует код JavaScript функции, которая выполняется на стороне клиента.

Вызов осуществляется конструкцией `#server(...)#`, где внутри скобок находится произвольный код на Caché ObjectScript. В нашем случае – вызов метода `..Convert`.

Таким образом, мы можем, во-первых, вызывать из браузера любую функцию на сервере, а, во вторых, инициировать на сервере Caché любые изменения на странице, используя JavaScript.

В Caché можно вызывать серверные методы с помощью Гипер-событий не только с помощью `#server(...)#`, но и с помощью `#call(...)#`. В этом случае на клиенте не нужен Java-апплет, но, важно отметить, что в отличие от `#server(...)#`, вызов серверных методов с помощью `#call(...)#` осуществляется асинхронно.

Более подробную информацию можно найти в документации (Документация → Developing Applications with Caché → Using Caché Server Pages (CSP) → Tag-Based Development with CSP → Server-Side Methods)

Поскольку внутри вызываемого кода Вы можете явным образом обращаться к объектной модели документа в браузере, комбинируя технологии Гипер-событий и Динамического HTML, то можно создавать пользовательские интерфейсы произвольной сложности, по функционалу ничем не отличающиеся от обычных Клиент - Серверных приложений, написанных, например, на Visual Basic или Delphi.

Привязка объектов к форме. Тег **CSP:OBJECT**, **CSPBIND**

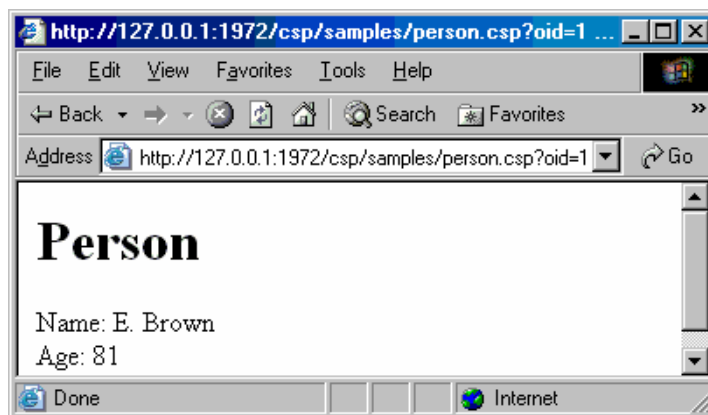
Мы уже выполнили примеры, в которых рассматривались операции поиска в БД, работы с запросами к БД и вывода данных из БД на экран. Для создания полноценных приложений необходимо научиться добавлять, изменять и удалять данные БД. У разработчика возникает необходимость связать объект Cache и CSP-страницу. Для этого в CSP предусмотрен специальный тег – CSP:Object .

Например: `<csp:object Name="obj" classname="Sample.Person" oid=1>`. Тег `<CSP:Object>` открывает объект. Первый параметр Name дает ему имя для использования на Web-странице, второй параметр ClassName устанавливает класс объекта, третий параметр oid - ID объекта внутри класса (если ObjId="", то создается новый объект).

Создайте файл person.csp следующего содержания:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<csp:OBJECT NAME="obj" CLASSNAME="Sample.Person"
OBJID=#($Get(%request.Data("oid",1))#>
<h1>Person</h1>
<p>
  Name: #(obj.Name)#<br>
  Age: #(obj.Age)#<br>
</p>
</BODY>
</HTML>
```

Загрузите в браузер страницу: <http://127.0.0.1:1972/csp/samples/person.csp?oid=1>



Попробуйте менять значение параметра oid, чтобы просмотреть значения других объектов базы.

Тег `<CSP:Object>` позволяет связать данные объекта с формой HTML через атрибут тега формы CSPBind.

Подробнее с возможностями тега CSPBind Вы можете ознакомиться в документации (Документация → Developing Applications with Caché → Using Caché Server Pages (CSP) → Building Database Applications → Binding Data to Forms)

С помощью CSPBind нужно связать объект и форму, а также поля формы и соответствующие атрибуты объекта. Например:

```
<csp:object name="obj" classname="Sample.Person" objid=1>
<form method cspbind=obj name=Person>
  Name: <input type=TEXT name="Name" cspbind=Name><br>
  Age: <input type=TEXT name="Age" cspbind=Age>
</form>
```

В форме будет показан экземпляр класса Sample.Person с id=1.

Если осуществляется привязка объекта к форме CSP-страницы тегом <CSP:Object>, Caché автоматически создает методы formname_new() и formname_save() (где formname – имя формы) для добавления новых объектов и сохранения изменений.

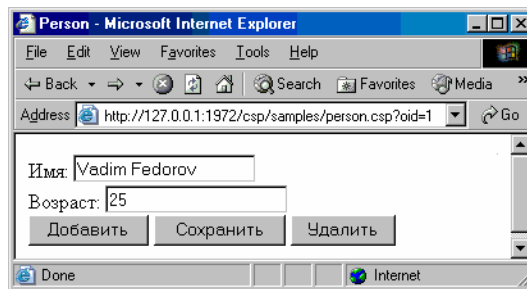
Если мы допишем метод Del(), который будет удалять объект из БД, то мы можем создать CSP-страницу, которая позволит пользователю осуществлять основные операции с экземплярами класса Sample.Person на CSP-странице.

Усовершенствуем Person.csp:

```
<HTML>
<HEAD><title>Person</title>
</HEAD>
<script language="Cache" runat="server">
  if '$Data(%session.Data("oid",1)){
    set %session.Preserve=1
    Set %session.Data("id",1)=$Get(%request.Data("oid",1))
  }
</script>
<BODY bgcolor="#FFFFFF">
<csp:object name="obj" classname="Sample.Person"
OBJID=#(%session.Data("id",1))#>
<form name=Person cspbind=obj action="PersonSearch.csp">
  Имя: <input type=TEXT name="Name" cspbind=Name><br>
  SSN: <input type=TEXT name="SSN" cspbind=SSN ><br>

  <input name="New" type="Button" value="Добавить" onClick='Person_new();'>
  <input name="Save" type="Button" value="Сохранить" onClick='Person_save();'>
  <input name="Del" type="Button" value="Удалить" onClick="#server(..Del())#">
</form>
</BODY>
</HTML>
<script language="Cache" Method="Del" Arguments="">
  do ##class(Sample.Person).%DeleteId($Get(%session.Data("id",1)))
  &javascript<
    self.document.Person.Name.value="";
    self.document.Person.SSN.value="";>
</script>
```

В результате получается следующая форма, которая позволяет Вам не только просматривать, но и редактировать объекты в базе:



Cache Web Form Wizard создает страницы, использующие механизм CSPBIND. Вы можете использовать автоматически порожденный код в качестве примера при изучении механизма CSPBIND.

Шифрование и безопасность

В созданном нами приложении практически отсутствовало какое бы то ни было обеспечение безопасности. Любой пользователь мог обратиться к произвольной странице нашего приложения и получить информацию о любом объекте. В CSP существуют встроенные технологии обеспечения безопасности, а именно, концепции PRIVATE и ENCODED страниц.

Рассмотрим параметры Private и Encoded объекта %CSP.Page.

CSP-страница может быть Public или Private. Если присвоить свойство Private=1 (по умолчанию Private=0), то на страницу можно будет попасть только с другой CSP-страницы.

Если у страницы Encoded=1 (по умолчанию Encoded=0), то все параметры при передаче зашифровываются и передаются с помощью CSPToken. В результате только зашифрованные параметры доступны через объект %CSP.Request.

Измените значения параметров страниц PersonSearch.csp и Person.csp.

Для этого нужно добавить в начало файлов следующий код:

```
<csp:class PRIVATE="1" ENCODED="1">
```

Теперь пользователь не может загрузить эти страницы через URL, и при переходе между PersonSearch.csp и Person.csp параметры зашифрованы.

После присваивания атрибутов PRIVATE или ENCODED какой либо странице, не забудьте перекомпилировать все страницы, которые на нее ссылаются. В нашем случае – это login.csp.

Заметьте, что URL CSP-страниц, которые открываются в браузере, заметно отличается оттого, что описывается в исходном коде CSP страницы. Для большинства стандартных тегов, например, или <form action="...">, компилятор CSP автоматически конвертирует url из внутреннего представления во внешнее. Но в некоторых случаях необходимо позаботиться об этом самостоятельно. Для этого Вы можете использовать директиву #url(myPrivatePage.CSP)#. Например – в коде JavaScript Вашей страницы вместо:

```
self.document.location='private.csp';
```

необходимо использовать:

```
self.document.location='#url(private.csp)#';
```

Внутри кода Cache Object Script Вы можете использовать вызов метода ..Link("private.csp").

Include

В Cache Server Pages есть Тег CSP:INCLUDE, который позволяет включить другую страницу во время исполнения.

Создадим страницу included.csp:

```
<p>
  Работа с #($get(%request.Data("PARAM",1))#
</p>
```

Создайте страницу Test.csp и откройте её в браузере:

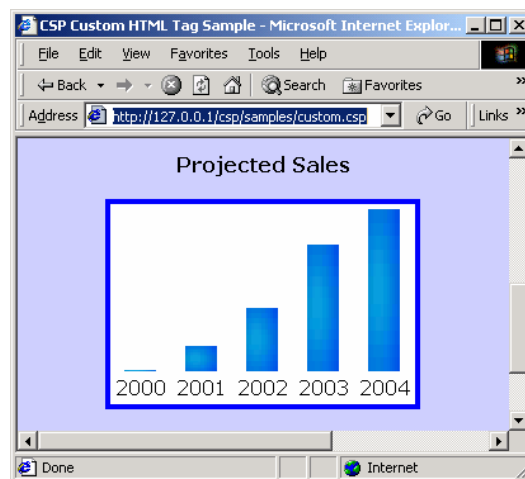
```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <csp:Include Page="included.csp?PARAM=Include" >
</BODY>
</HTML>
```

Более подробную информацию по тегу <CSP:INCLUDE> можно найти в документации (Документация → Reference Material → CSP HTML Tag Reference → <CSP:INCLUDE>).

В Caché Server Pages существует и другой тег #INCLUDE, позволяющий вставлять в страницы код из других документов, который будет использоваться при компиляции (Документация → Reference Material → CSP HTML Tag Reference → #INCLUDE).

Создание собственных тегов

Один из наиболее мощных механизмов, заключенных в CSP – возможность создания собственных тегов (custom tags). Загрузите в браузер пример страницы, использующей собственный тег: <http://127.0.0.1:1972/csp/samples/custom.csp>



Обратите внимание на то, насколько просто при этом выглядит исходный код той части CSP-файла, что отвечает за вывод диаграммы:

```
<isc:BARChart BGCOLOR="white">
  <isc:ELEMENT HEIGHT="1" LABEL="2000">
  <isc:ELEMENT HEIGHT="20" LABEL="2001">
  <isc:ELEMENT HEIGHT="50" LABEL="2002">
  <isc:ELEMENT HEIGHT="100" LABEL="2003">
  <isc:ELEMENT HEIGHT="#"(100+$Random(200))#" LABEL="2004">
</isc:BARChart>
```

Тегов <isc:BARChart ...> и <isc:ELEMENT ...> нет в стандартном наборе тегов HTML. Они созданы в CSP с использованием технологии пользовательских тегов.

Создадим собственный тег самостоятельно. Для этого мы должны создать Caché Server Rule (CSR). Создайте в Caché Studio новую CSP-страницу.

```
<csr:RULE name="TODAY" match="TODAY" empty>
```

```
<csr:ACTION>
Сегодня: <b>#( $ZDATE( $H) )#</b>
</csr:ACTION>
</csr:RULE>
```

Сохраните CSR как Today.csr (Обратите внимание, что расширение у файла на CSP, а CSR).

Откомпилируйте страницу в Caché Studio или в терминале наберите следующую команду:

```
SAMPLES>do $system.CSP.LoadRule("c:\cachesys\csp\samples\today.csr")
```

Измените CSP-страницу Test.CSP:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
    <csp:Include Page="included.csp?PARAM=Include">
    <TODAY>
</BODY>
</HTML>
```

Загрузите страницу Test.csp в браузер.

Подробное руководство по созданию собственных CSP-тегов Вы найдете в документации (Документация → Developing Applications with Caché → Using Caché Server Pages (CSP) → Developing Custom Tags). Перед изучением механизма создания пользовательских тегов будет полезно просмотреть исходный код тегов, входящих в поставку Caché, таких как <csp:if ...>, <csp:search ...> и других. Вы найдете их в каталоге C:\CacheSys\Dev\csp\rules. Вы можете создать свои собственные теги на базе стандартных. Например, можно создать тега <csp:search ...> с русскоязычным интерфейсом.

Что дальше?

Если вы успешно прошли все этапы этого курса, то Вы уже можете создавать достаточно сложные приложения Caché Server Pages. Однако, возможности CSP далеко не исчерпываются рассмотренным. При помощи документации и дополнительных источников информации, указанных в [Приложении 3](#), рекомендуем Вам самостоятельно ознакомиться со следующими аспектами разработки Web приложений:

- Конфигурирование CSP
- Режим работы %session.Preserve=1
- Управление лицензиями
- Использование различных элементов формы (Radio, Select и пр.) в CSP приложениях
- Хранение изображений в базе данных
- Использование Cookie, переменных CGI и управление заголовками HTTP
- Вызов серверных методов с помощью #server и #call
- Создание собственных тегов
- Создание многоязычных CSP-интерфейсов с помощью CSP Localization.
- Работа с XML
- Работа с CSP-приложением DOCBOOK (документация Caché)

Приложение 1. Советы по отладке CSP-приложений

✓ При построении CSP-приложений наиболее часто встречаются ошибки, связанные с использованием кавычек. Например, такая простая конструкция как:

```
<a href="person.csp?ObjID=(query.Get("ID"))#">
```

Вызовет ошибку, а

```
<a href='person.csp?ObjID=#(query.Get("ID"))#'>
```

уже будет интерпретирована корректно. В первом случае компилятор воспримет весь код внутри первой пары двойных кавычек `person.csp?ObjID=#(query.Get(` как значения параметра, что, естественно приведет к ошибке. Поэтому везде, где возможна некорректная интерпретация кавычек, используйте одинарные кавычки в коде HTML и двойные в коде, который будет интерпретироваться как Caché Script.

✓ Вы можете отлаживать CSP в терминале:

```
do $system.CSP.Shell()
```

✓ Внутри страницы Вы можете установить параметр, который выведет содержание объекта `%request` после того, как вывод Вашей страницы будет завершен.

```
Set %response.TraceDump=1
```

Пример использования `%response.TraceDump` - на странице <http://127.0.0.1:1972/csp/samples/inspector.csp>

✓ Включите в Caché Studio опцию «Сохранять порожденный исходный код». В этом случае по INT коду вы сможете обнаружить точное место появления ошибки.

Приложение 2. Интеграция с Macromedia DreamWeaver.

Вы можете создавать CSP-страницы с использованием произвольного редактора HTML-страниц. При использовании Macromedia DreamWeaver Вы можете использовать специально созданные для DreamWeaver расширения для работы с тегами CSP.

Более подробную информацию и по настройке, и по использованию Macromedia DreamWeaver Вы можете найти в документации (Документация → Developing Applications with Caché → Using Caché Server Pages (CSP) → Using CSP with Macromedia Dreamweaver)

Приложение 3. Дополнительные источники информации по CSP

Документация по Caché содержит следующую информацию по Caché Server Pages:

- Раздел документации «Using Caché Server Pages» (Документация → Developing Applications with Caché → Using Caché Server Pages (CSP)), который содержит подробную документацию по Caché Server Pages и учебное пособие Building Web Applications With Caché для самостоятельного изучения.
- Описание всех CSP-тегов «CSP HTML Tag Reference» (Документация → Reference Material → CSP HTML Tag Reference)
- Раздел «Knowledge Base» документации Caché, посвященный CSP (Документация → Knowledge Base → CSP)
- Статья «License Management in Caché 5» (Документация → Knowledge Base → System → License Management in Caché 5), которая содержит информацию об управлении лицензиями в Caché.

Полезными будут следующие ссылки:

- <http://127.0.0.1:1972/csp/samples/rulemgr.csp> - список всех загруженных в систему CSP-тегов.
- <http://127.0.0.1:1972/csp/samples/menu.csp> - примеры CSP-страниц с исходными кодами.
- <http://127.0.0.1:1972/apps/documatic> - документация по системным классам Caché. Базовые классы CSP находятся в пакете `%CSP`. Внимательно посмотрите на классы `%CSP.Page`, `%CSP.Session`, `%CSP.Request`, `%CSP.Response`, `%CSP.Error`

Следующая команда выведет список команд класса `$$SYSTEM.CSP`: `do $system.CSP.Help()`

Если Вам нужны дополнительные материалы по CSP, обращайтесь в [InterSystems Corporation](http://www.intersystems.com).