# APIBridge Plug-in API

## Package Description

The APIBridge plug-in creation package includes the following components:

- **APIBridge_Emulator.zip**: This file contains the binaries and header files needed to set up the development environment for creating plug-ins.
- **Examples**: This directory contains sample plug-in code that can be used as a starting point for any plug-in. The EchoServlet sample is used in this document to illustrate the steps for creating a plug-in.

## Setting Up the Development Environment

To create a plug-in for the APIBridge, the following tools are required:

- S60 SDK (3rd Edition or later)
- Carbide.c++

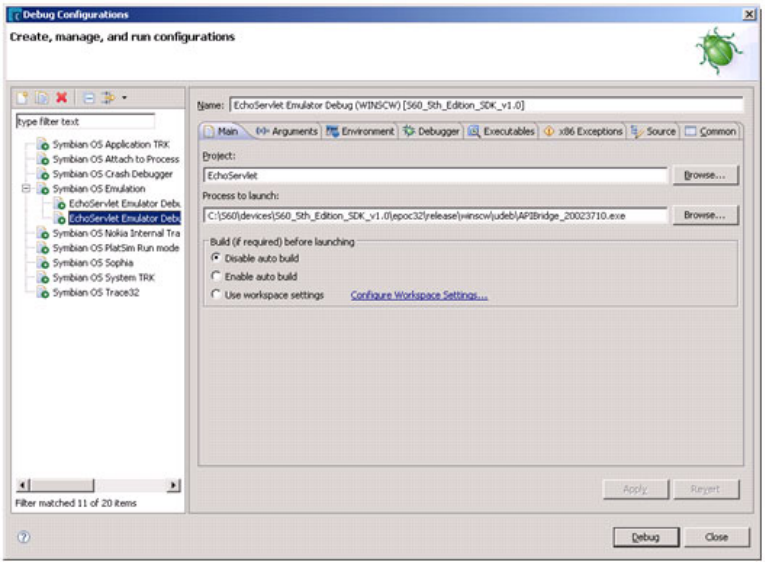
The steps outlined below should be followed to set up the development environment.

### Add the APIBridge files to your SDK

- Locate the root directory for the SDK. This is the one containing the epoc32 directory (for example, C:\S60\devices\S60_5th_Edition_SDK_v1.0).
- Unzip APIBridge_Emulator.zip in this directory.

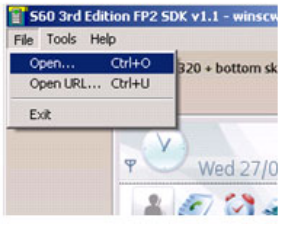
### Build the EchoServlet plug-in

- Using Carbide.c++, import the project contained in the \Examples\EchoServlet directory.
- Add the SDK you chose in the previous step to the Build Configurations.
- Build the software.
- Create a new Debug Configuration for the project and set the Process to Launch to: <SDK Directory>\epoc32\release\winscw\udeb\APIBridge_20023710.exe.




### Test the plug-in

- Launch the project in Debug using the Debug configuration created in the previous step.
- When the emulator has finished loading, install the EchoTest.wgz located in Examples\EchoServlet\test. Do this by opening the file from the emulator file menu.



- Launch the test application by navigating to it inside the emulator: Menu -> Applications -> EchoTest.

# Anatomy of a Plug-in

A plug-in is composed of two files:

- The ECOM resource file that tells the framework which requests is the plug-in going to serve
- The Servlet DLL, which implements the code necessary to fulfill the request.

Any plug-in requires the use of two or more UIDs:

- The UID for the DLL (It's called <DLL UID> in the example);
- The UID(s) for each of the request types serviced by the plug-in (It's called <request UID> in the example).

In the Example directory of the package you will find the EchoServlet plug-in used in this section.

## ECOM resource file

The source for this can be found in \data\20024644.rss. This resource file defines which requests are served by the plug-in.

```
#include <RegistryInfo.rh>

// Declares info for implementations
RESOURCE REGISTRY_INFO theInfo
{
 // UID for the DLL. Must me same as in mmp file
 dll_uid = <DLL_UID>; //Change this with your dll UID
 // Declare array of interface info
 interfaces =
  {
  INTERFACE_INFO
   {
   // UID of interface that is implemented. UID of the ApiBridge interface
   // DO Not Change
   interface_uid = 0x20023711;
   implementations =
    {
    IMPLEMENTATION_INFO
     {
     //The UID of your implementation
// same as in proxy.cpp implementation
     implementation_uid = <request UID>;
   version_no = 1;
     display_name = "Sample Echo Servlet";
     //This is the url that this servlet will serve
     default_data = "/sample/echo";
     }
    };
   }
  };
}
```

The three areas that are highlighted above are:

- **DLL UID**: Tells the system which DLL implements this interface. Make sure that the UID used here is the same as the UID indicated in your DLL's .mmp file.

```
TARGET EchoServlet_20024644.dll
TARGETTYPE PLUGIN
UID 0x10009D8D
```

- **Implementation UID**: Tells the system how to find the class that would take care of the request. Make sure that this UID is specified in your proxy.cp file:
  `IMPLEMENTATION_PROXY_ENTRY(<request uid>, CEchoServlet::NewL)`
- **Default Data**: A string that defines which request URL this plug-in will serve.

The `IMPLEMENTATION_INFO` block can be repeated as many times as requests are served by the plug-in.

## Servlet object

The servlet object will be responsible for creating the instance that will process the request. In the EchoServlet example, it can be found in the \src\EchoServlet.cpp file. Mainly, two functions are required. For more information, see the API description.

- **NewL function**: The function responsible for creating the object. It is called by the framework when it has been detected that a client has made a request which the plug-in is registered to process. Your proxy file will make this connection.
  `CEchoServlet* CEchoServlet::NewL()`
- **ServiceL function**: The function that is the entry point for each request, and it passes the request object as a parameter. The request is completed by using the request object passed.

**void CEchoServlet::ServiceL**( MHttpRequest* req )

## ECOM proxy file

The proxy file can be found in \src\proxy.cpp in the sample project. Its function is to link the request types with the C++ object that will process it.

```
const TImplementationProxy ImplementationTable[] =
  {IMPLEMENTATION_PROXY_ENTRY(<request uid>, CEchoServlet::NewL) };
```

There can be as many entries in this table as requests processed by this plug-in.

### Installation file

There are two files to copy into a device when installing a plug-in: the ECOM resource file in c:\resource\plugins and the dll file in c:\sys\bin. Also the dependency with the APIBridge needs to be stated. The .pkg for the sample can be found in the \sis directory.

```
;Dependency with APIBridge
[0x2002373F],1, 0, 6,{"APIBridge"}
;Servlet files
"EchoServlet_20024644.rsc" - "C:\Resource\Plugins\EchoServlet_20024644.rsc"
"\EchoServlet_20024644.dll" - "C:\sys\bin\EchoServlet_20024644.dll"
```

### Binding code

For the plug-in to be accessible from the different runtimes, binding code needs to be created. See the following pages:

- Creating APIBridge JavaScript Binding Code
- Binding code for Flash from Adobe (coming soon)
- Binding code for Java™ (coming soon)

## APIBridge Plug-in API

This chapter describes the Symbian API available for APIBridge plug-ins.

### CServlet class

This is the base class for all servlets; it controls the entry point into the plug-in and the security.

#### ServiceL function

```
virtual void ServiceL ( MHttpRequest* req ) = 0
```

This function must be implemented by the plug-in. It is the entry point into the plug-in code and receives the HTTP request from the binding code.

| Parameter | Description |
|---|---|
| **req** | HTTP request object containing both the request made by the binding code and the response object. |

#### GetSecurityType function

```
TSecurityType GetSecurityType ()
```

The implementation of this function is optional. It informs the framework about which security mechanism would be required by the plug-in.

| Parameter | Description |
|---|---|
| **return** | There are three possible values:<br>- *ENone:* The plug-in doesn't require security; any client can use it.<br>- *EPrompt:* The plug-in is secured by a prompt to the user. The prompt authenticates a new application trying to access the plug-in. This is the default implementation.<br>- *ESecure:* The plug-in is secured by implementing the **IsClientValid** function. No prompt will be shown to the user. |

#### IsClientValid function

```
TBool IsClientValid(const TDesC8& hash )
```

This function must be implemented only if the security mode is set to Secure.

| Parameter | Description |
|---|---|
| **hash** | An md5 of the code files in the calling app. This can be used for reference in order to authenticate one application only. |
| **return** | - True: The request will be allowed to go through.<br>- False: The request will be denied. |

### MCookie class

This class represents a cookie in the HTTP request or response.

#### GetName function

```
TPtrC8 GetName()
```

This function returns the name of the cookie. The name cannot be changed after creation.

| Parameter | Description |
|---|---|
| **return** | String containing the name of the cookie. |

#### GetValue function

```
TPtrC8 GetValue()
```

This function returns the value of the cookie.

| Parameter | Description |
| --- | --- |
| return | String containing the cookie's present value. |

### GetVersion function

```
TInt GetVersion()
```

This function returns the version of the protocol with which this cookie complies. Version 1 complies with RFC 2109, and version 0 complies with the original cookie specification drafted by Netscape. Cookies provided by a browser use and identify the browser's cookie version.

| Parameter | Description |
| --- | --- |
| return | 0 if the cookie complies with the original Netscape specification; 1 if the cookie complies with RFC 2109. |

### SetVersion function

```
void SetVersion (TInt v)
```

This function sets the version of the cookie protocol to which this cookie complies. Version 0 complies with the original Netscape cookie specification. Version 1 complies with RFC 2109. Since RFC 2109 is still somewhat new, consider version 1 to be experimental; do not use it yet on production sites.

| Parameter | Description |
| --- | --- |
| v | 0 if the cookie should comply with the original Netscape specification; 1 if the cookie should comply with RFC 2109. |

### GetPath function

```
TPtrC8 GetPath()
```

This function returns the path on the server to which the browser returns this cookie. The cookie is visible to all subpaths on the server.

| Parameter | Description |
| --- | --- |
| return | A string specifying a path that contains a servlet name, for example, /catalog. |

### SetPathL function

```
SetPathL(const TDesC8& uri)
```

This function specifies a path for the cookie to which the client should return the cookie. The cookie is visible to all the pages in the directory you specify and all the pages in that directory's subdirectories. A cookie's path must include the servlet that set the cookie, for example, /catalog, which makes the cookie visible to all directories on the server under /catalog.

Consult RFC 2109 (available on the internet) for more information on setting path names for cookies.

| Parameter | Description |
| --- | --- |
| uri | A string specifying a path. |

### GetMaxAge function

```
TInt GetMaxAge()
```

This function returns the maximum age of the cookie, specified in seconds; by default, -1 indicates the cookie will persist until browser shutdown.

| Parameter | Description |
| --- | --- |
| return | An integer specifying the maximum age of the cookie in seconds; if negative, means the cookie persists until browser shutdown. |

### SetMaxAge function

```
void SetMaxAge(TInt expiry)
```

This function sets the maximum age of the cookie in seconds. A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the maximum age when the cookie will expire, not the cookie's current age. A negative value means that the cookie is not stored persistently and will be deleted when the web browser exits. A 0 value causes the cookie to be deleted. Returns the maximum age of the cookie, specified in seconds; by default, -1 indicates the cookie will persist until browser shutdown.

| Parameter | Description |
| --- | --- |
| expiry | An integer specifying the maximum age of the cookie in seconds; if negative, means the cookie is not stored; if 0, deletes the cookie. |

## MHttpRequest class

This class represents the HTTP request and response objects.

### GetRequest function

```
MHttpServletRequest* GetRequest()
```

This function provides the request object, which contains the information that was initiated at the binding layer.

| Parameter | Description |
| --- | --- |
| return | Request object. |

### GetResponse function

```
MHttpServletResponse* GetResponse()
```

This function provides the response object, which contains the information that has to be populated and sent back to the binding layer.

| Parameter | Description |
|---|---|
| **return** | Response object. |

### CreateCookieLC function

```
CreateCookieLC(const TDesC8& name, const TDesC8& value )
```

This is a helper function that allows an MCookie object to be created.

| Parameter | Description |
|---|---|
| **Name** | Name of the cookie. |
| **Value** | Value of the cookie. |
| **return** | Cookie object. |

## MHttpServletRequest class

This class represents the HTTP request object, which contains the information that was passed to the plug-in by the binding layer.

### GetHeader function

```
TPtrC8 GetHeader(const TDesC8& name )
```

This function returns the value of the specified request header as a string. If the request did not include a header of the specified name, this method returns null. The header name is case insensitive. You can use this method with any request header.

| Parameter | Description |
|---|---|
| **name** | String specifying the header name. |
| **return** | String containing the value of the requested header, or null. |

### GetIntHeaderL function

```
TInt GetIntHeaderL(const TDesC8& name )
```

This function returns the value of the specified request header as an integer. If the request does not have a header of the specified name, this method returns -1. If the header cannot be converted to an integer, this method leaves.

| Parameter | Description |
|---|---|
| **name** | String specifying the name of a request header. |
| **return** | Integer expressing the value of the request header or -1 if the request doesn't have a header of this name. |

### GetHeadersL function

```
void GetHeadersL(const TDesC8& name, RStringCollection& values )
```

This function returns all the values of the specified request header as an enumeration of string objects.

| Parameter | Description |
|---|---|
| **name** | String specifying the header name. |
| **values** | Enumeration containing the values of the requested header, or null if the request does not have any headers of that name. |

### GetHeaderNamesL function

```
void GetHeaderNamesL ( RStringCollection& names )
```

This function returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration.

| Parameter | Description |
|---|---|
| **names** | Enumeration of all the header names sent with this request; if the request has no headers, an empty enumeration. |

### GetMethod function

```
TPtrC8 GetMethod()
```

This function returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.

| Parameter | Description |
|---|---|
| **return** | String specifying the name of the method with which this request was made. |

### GetPath function

```
TPtrC8 GetPath()
```

This function returns the portion of the request URI that indicates the context of the request. The context path always comes first in a request URI. The path starts with a '/' character but does not end with a '/' character. For servlets in the default (root) context, this method returns "".

| Parameter | Description |
|---|---|
| return | String specifying the portion of the request URI that indicates the context of the request. |

### GetQuery function

```
TPtrC8 GetQuery()
```

This function returns the query string that is contained in the request URL after the path. The method returns an empty string if the URL does not have a query string.

| Parameter | Description |
|---|---|
| return | String containing the query string or empty descriptor if the URL contains no query string. |

### GetContent function

```
TPtrC8 GetContent()
```

This function returns the content of the request body.

| Parameter | Description |
|---|---|
| return | String containing the content of the body or empty descriptor if the URL contains no query string. |

### GetCookiesL function

```
void GetCookiesL( RCookieCollection& cookies )
```

This function returns an array containing all of the cookie objects the client sent with this request. The method returns an empty array if no cookies were sent.

| Parameter | Description |
|---|---|
| return | Array of all the cookies included with this request, or an empty array if the request has no cookies. |

## MHttpServletResponse class

This class represents the HTTP response object that contains the information that will be passed to the binding layer by the plug-in.

### AddCookieL function

```
void AddCookieL( const MCookie& cookie )
```

This function adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.

| Parameter | Description |
|---|---|
| cookie | The cookie object to be added to the response. |

### AddHeaderL function

```
void AddHeaderL( const TDesC8& name, const TDesC8& value )
```

This function adds a response header with the given name and value. The method allows response headers to have multiple values.

| Parameter | Description |
|---|---|
| name | String containing the name of the HTTP header. |
| value | String containing the value of the HTTP header. |

### AddIntHeaderL function

```
void AddIntHeaderL ( const TDesC8& name, TInt value )
```

This function adds a response header with the given name and integer value. The method allows response headers to have multiple values.

| Parameter | Description |
|---|---|
| name | String containing the name of the HTTP header. |
| value | Integer containing the value of the HTTP header. |

### ContainsHeader function

```
TBool ContainsHeader( const TDesC8& name )
```

This function returns a Boolean indicating whether the named response header has already been set.

| Parameter | Description |
|---|---|
| name | String containing the name of the HTTP header. |
| return | True if the named response header has already been set; false otherwise. |

### SetHeaderL function

```
void SetHeaderL(const TDesC8& name, const TDesC8& value )
```

This function sets a response header with the given name and value. If the header had already been set, the new value overwrites the previous one. The containsHeader method can be used to test for the presence of a header.

| Parameter | Description |
| --- | --- |
| **name** | String containing the name of the HTTP header. |
| **value** | String containing the value of the HTTP header. |

### SetIntHeaderL function

```
void SetIntHeaderL( const TDesC8& name, TInt value )
```

This function sets a response header with the given name and integer value. If the header had already been set, the new value overwrites the previous one. The containsHeader method can be used to test for the presence of a header before setting its value.

| Parameter | Description |
| --- | --- |
| **name** | String containing the name of the HTTP header. |
| **value** | Integer containing the value of the HTTP header. |

### SetStatus function

```
void SetStatus( TInt sc )
```

This function sets the status code for this response. This method is used to set the return status code when there is no error. If there is an error, the sendError method should be used instead.

| Parameter | Description |
| --- | --- |
| **sc** | Status code. |

### SendErrorL function

```
void SendErrorL( TInt sc )
```

This function sends an error response to the client using the specified status. The server generally creates the response to look like a normal server error page.

| Parameter | Description |
| --- | --- |
| **sc** | Error status code. |

```
void SendErrorL( TInt sc, const TDesC8& msg )
```

This sends an error response to the client using the specified status code and descriptive message. The server generally creates the response to look like a normal server error page.

| Parameter | Description |
| --- | --- |
| **sc** | Error status code. |
| **msg** | Error description message. |

### SendL function

```
void SendL( const TDesC8& content )
```

This function sends a response to the client with specified content.

| Parameter | Description |
| --- | --- |
| **content** | A Symbian description containing the body of the message to be sent to the binding code. |

```
void SendL( const TUint8* ptr, TInt length )
```

This sends a response to the client with specified content.

| Parameter | Description |
| --- | --- |
| **ptr** | Pointer to the content to be included within the body of the message. |
| **length** | Length of the content. |

### SetListener function

```
void SetListener( MHttpResponseListener* listener )
```

This function assigns a listener object to the request. This will be used to get informed when the request is done.

| Parameter | Description |
| --- | --- |
| **listener** | Listener object to listen for this request. |

## MHttpResponseListener class

This class represents a listener object that is used to get notifications regarding the completion of the request.

### OnSendDone function

```
virtual void OnSendDone() = 0
```

This function needs to be implement by the object and gets called when the request has been sent to the binding layer. The object needs to be added to the response object via the SetListener function in the MHttpServletResponse object.

| Parameter | Description |
|---|---|
| **cookie** | The cookie object to be added to the response. |

## RQueryParser class

This is a helper class used to parse URI parameters.

### ParseL function

```
void ParseL(const TDesC8& query )
```

This function initialises the object by ingesting the query string.

| Parameter | Description |
|---|---|
| **query** | String containing the query as received by the binding layer. |

### GetValue function

```
TPtrC8 GetValue( const TDesC8& name )
```

This function retrieves the value for a URI parameter.

| Parameter | Description |
|---|---|
| **name** | String containing the name of the parameter. |
| **return** | String containing the value of the parameter. |

### HasValue function

```
TBool HasValue( const TDesC8& name )
```

This function informs us of the existence of a parameter.

| Parameter | Description |
|---|---|
| **name** | String containing the name of the parameter. |
| **return** | True if the parameter exists or false if it doesn't. |

### Count function

```
TInt Count()
```

This function returns the number of parameters in the query.

| Parameter | Description |
|---|---|
| **return** | Number of parameters in the query. |

### GetName function

```
TPtrC8 GetName(TInt idx)
```

This function returns the name of the parameter in position idx.

| Parameter | Description |
|---|---|
| **idx** | Position. |

### GetValue function

```
TPtrC8 GetValue(TInt idx)
```

This function returns the value of the parameter in position idx.

| Parameter | Description |
|---|---|
| **idx** | Position. |

### Close function

```
void Close()
```

This function closes the object and destroys all internal structures. It must be called when the object is no longer needed.