SCO Forum 2006

MOBILITY EVERYWHERE >

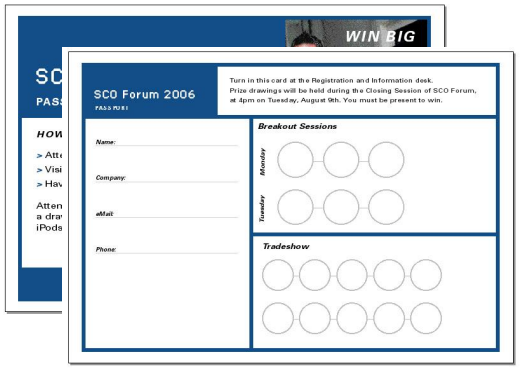
Building EdgeClick clients for Windows Mobile smartphones with .NET

Michael Almond

Session ID: 129

Get Your Passport Stamped

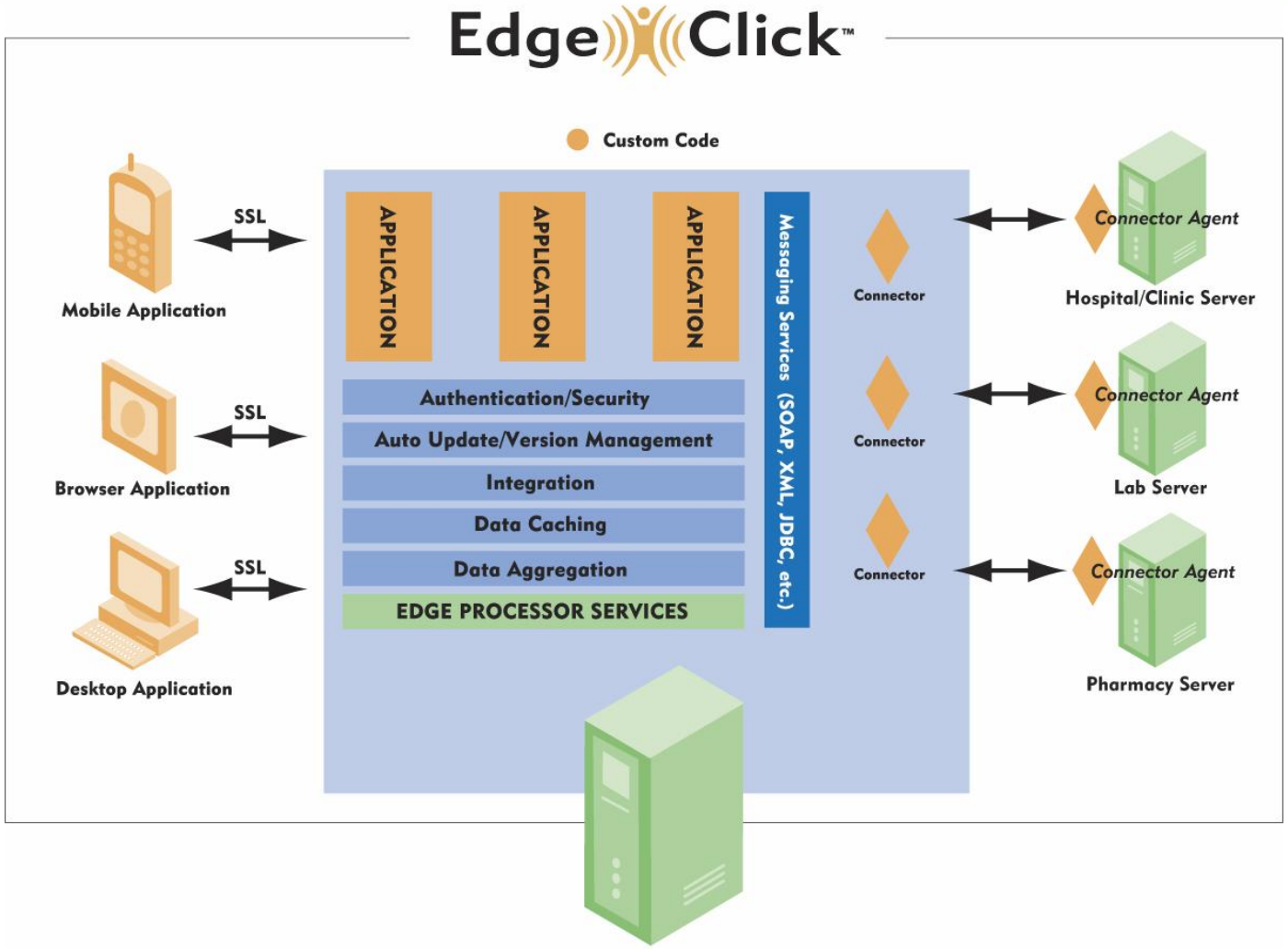
- Be sure to get your Passport stamped.
  - Get your passport stamped
    - By breakout session instructors
    - By exhibitors in the exhibit hall
  - Turn in your Passport
    - After the last breakout session on Wednesday
    - Drawing for great prizes for Wrap-up Session
- Remember to complete the breakout session evaluation form, too



Agenda

- EdgeClick Architecture
- Development Environment
- EdgeClick .NET API
- Constructing a .NET EdgeClick Client
- Packaging a .NET EdgeClick Client
- Development Tips & Tricks

EdgeClick Architecture



Development Environment

- Windows Mobile clients are written in Visual C#, using the .NET 2.0 Compact Framework.
- Requirements:
  - ActiveSync 4.1
  - Visual Studio 2005 (Standard edition or better)
  - Windows Mobile 5.0 SDK
  - EdgeBuilder SDK (includes the EdgeClick .NET API)
  - Windows Mobile PocketPC-based smart device

EdgeClick .NET API – Components

- The EdgeClick .NET 2.0 API provides a set of .NET assemblies to authenticate and communicate with the EdgeClick processor:
  - EdgeClick.Accounts.dll – manipulate and authenticate accounts
  - EdgeClick.Groups.dll – manipulate subscriber groups and contacts
  - EdgeClick.Net.dll – send/receive HTTP requests and encode/ decode URLs
  - EdgeClick.Utility.dll – a variety of utilities, including safe data type parsing, phone information, access SQLite databases, and error handling
- It also includes SQLite, a lightweight SQL ADO.NET database provider, used to store persistent data on the phone.

EdgeClick .NET API – Namespaces

- EdgeClick.Accounts
  - **Account** – class representing a single account
  - **AccountManager** – class providing user input forms for creating, editing, deleting and authenticating accounts
  - **AppMainForm** – class providing standard EdgeClick menu items and event handlers, plus initializes the AccountManager
- EdgeClick.Groups
  - **Contact** – class representing a single contact
  - **Group** – class representing a single group
  - **GroupManager** – class providing user input forms for creating, editing, deleting and synchronizing groups and contacts
  - **frmContacts** – form to select contacts from a list
  - **frmGroups** – form to select groups from a list

EdgeClick .NET API – Namespaces

- EdgeClick.Net
  - **Http** – class for sending and receiving HTTP requests
  - **HttpUtility** – class for encoding and decoding URLs
- EdgeClick.App
  - class that provides application properties that are required to be defined by every EdgeClick application and creates the single instances of the AccountManager, GroupManager and PleaseWaitBox classes

EdgeClick .NET API – Namespaces

- EdgeClick.Utility
  - **CustomListView** – class to extend ListView by providing a count of the number of checked items
  - **Database** – class for manipulating an SQLite database
  - **EdgeClick*Exception** – set of classes providing exceptions specific to error responses from the EdgeClick Processor
  - **Memory** – class for allocating and freeing memory on a Windows Mobile device
  - **NewStringBuilder** – class to provide easy methods to build strings (especially SQL queries)
  - **Phone** – class that wraps the Pocket PC Phone API (make a call, get the call log)
  - **PleaseWaitBox** – class that displays a "Please wait" box with a message and turns on/off the wait cursor
  - **Utilities** – a collection of useful functions, including safe data type parsing and error handling

EdgeClick .NET API – AccountManager

- Only one instance of the **EdgeClick.Accounts.AccountManager** class can exist, and is referenced via **App.AccountMgr**.
- Responsible for maintaining the account database and authenticating accounts.
- Provides forms for adding, modifying, deleting and authenticating accounts.
- Launched before the rest of your client application and will check if an authenticated account already exists in the account database.
- If no authenticated accounts are found, it will present the user with a form to create and authenticate an account with the EdgeClick Processor.
- Although multiple accounts can be created and authenticated, only one can be active (or current) at one time.
- **App.AccountMgr.CurrentAccount** will point to the current, authenticated account.

EdgeClick .NET API – GroupManager

- Only one instance of the **EdgeClick.Groups.GroupManager** class can exist, and is referenced via **App.GroupMgr**.
- Maintains the group and contact database on the phone.
- Two types of contacts, standard contacts that are in the phone's phonebook and web contacts that are stored on the EdgeClick Processor.
- Synchronizes the database with the EdgeClick Processor (automatically or manually).
- Provides forms to add, modify and delete groups and contacts.
- Provides forms for selecting groups or contacts for use in your client application as recipient lists.

Constructing a .NET EdgeClick Client

1. Create a new Visual C# Smart Device project
2. Add references to the EdgeClick .NET API to the project
3. Define some required EdgeClick properties for your application (name, version number, etc)
4. Inherit the AccountManager to authenticate and manage EdgeClick accounts
5. Add some EdgeClick menu items, for handling EdgeClick-specific items such as accounts and groups
6. Use the EdgeClick .NET API to send commands to the EdgeClick processor
7. Parse the results
8. Error handling
9. Version control

Step 1 – Create Project

- Create a new Visual C# Smart Device project:
  - Visual C# → Smart Device → Pocket PC 2003 → Device Application
  - Visual C# → Smart Device → Windows Mobile Pocket PC → Device Application
  - **NOTE:** the other Smart Device project types are not suitable for EdgeClick
- An EdgeClick template is also available as part of the EdgeBuilder SDK:
  - Visual C# → My Templates → EdgeClick Client
- The sample client included with EdgeBuilder can also be used as a starting point:
  - C:\EdgeBuilder\sdk\client\DotNetCF2\Samples\Sample

Step 2 – Reference EdgeClick API

- Add references to the EdgeClick assemblies to the project from the EdgeBuilder SDK at *C:/EdgeBuilder/sdk/client/DotNetCF2*

  EdgeClick.Utility.dll
  EdgeClick.Net.dll
  EdgeClick.Accounts.dll
  EdgeClick.Groups.dll – *if your app utilizes groups/contacts*
  System.Data.SQLite.dll – *if your app needs an SQL database*

- Add using references for the EdgeClick namespaces to your code:

```
using EdgeClick;
using EdgeClick.Utility;
using EdgeClick.Net;
using EdgeClick.Accounts;
using EdgeClick.Groups;
using System.Data.SQLite;
```

Step 3 – Define Application Properties

- In *Program.cs*, define a number of EdgeClick application properties which uniquely identifies your application to the EdgeClick Processor:

```
// Application's unique code number
EdgeClick.App.ApplicationIndex = "0";

// Application's name
EdgeClick.App.Name = "Sample";

// Application's version
EdgeClick.App.Version = "1.0.0";

// Application's brand name (shown on About screen)
EdgeClick.App.Brand = "EdgeClick";

// Application's marketing URL (shown on About scrn)
EdgeClick.App.URL = "www.edgeclickpark.com";
```

Step 3 – Define Application Properties

- You can override the new account default values that are used for defining the URL to the EdgeClick Processor.
- Specify new values in *Program.cs* for the **App.DefaultURI** properties:

```
// Use http instead of https
App.DefaultURI.PROTOCOL = "http://";
// Community name
App.DefaultURI.PAVILION = "";
// Domain name
App.DefaultURI.ENDPOINT = "";
// Application URL
App.DefaultURI.APPURL = "eps-1.0/epsmob";
```

Step 4 – Inherit AccountManager

- Your application's main form should inherit from AppMainForm which will result in executing the AccountManager before your application even starts:

```
public partial class myMainForm :
                                EdgeClick.Accounts.AppMainForm
```

- When the AccountManager executes it will try to load the account marked as current from the Account database on the phone.
- If it can't find a current account, then it will display the account form so the user can create a new account. The user will not be able to proceed until an account is successfully authenticated.
- **App.AccountMgr.CurrentAccount** will contain the current, authenticated account.
- Once an authenticated account is found or created, control will be returned to your application's main form, allowing its load and show routines to be executed, etc.

Step 5 – EdgeClick Menu

- It is recommended that all EdgeClick applications have an EdgeClick menu, with the following entries:

- only if your app utilizes groups
- only if your app utilizes contacts

- not implemented yet

- Button click event handlers for most of these menu items are inherited from AppMainForm:

```
accountsMenu.Click += new System.EventHandler(accountMenu_Click);
switchToMenu.Click += new System.EventHandler(switchToMenu_Click);
aboutMenu.Click += new System.EventHandler(aboutMenu_Click);
exitMenu.Click += new System.EventHandler(exitMenu_Click);
```

Step 5 – EdgeClick Menu

- You will to need to write button click event handlers for the Groups menu item:

```
this.myGroupsMenu.Click +=
          new System.EventHandler(this.groupMenu_Click);


/// <summary>
/// Display the Groups form
/// </summary>
private void groupMenu_Click(object sender, EventArgs e)
{
      frmGroups f = new frmGroups(false);
      f.ShowDialog();
}
```

Step 5 – EdgeClick Menu

- You will to need to write a button click event handler for the Contacts menu item:

```
this.myContactsMenu.Click +=
          new System.EventHandler(this. contactsMenu_Click);


/// <summary>
/// Display the Contacts form
/// </summary>
private void contactsMenu_Click(object sender, EventArgs e)
{
     frmContacts f = new frmContacts(false);
     f.ShowDialog();
}
```

Step 6 – Send Commands To EP

- Communication between the phone and the EdgeClick Processor is done via HTTP POST request.
- The results are returned as XML (**XmlNode**).
- Commands are identified via a unique integer code.
- Any additional data specific to that command can be specified via attribute/value pairs, constructed using **EdgeClick.Net.Http**.
- When sending a command to the EdgeClick Processor, always also use **App.PleaseWait** to display a box telling the user that communication is being performed over the network.

Step 6 – Send Commands To EP

- **App.SendCommandToEP** sends the specified command to the EdgeClick Processor (app code, version, and account information are automatically added to your command).
- **App.SendCommandToEP** will throw an exception so must be wrapped in a try/catch:

```
try
{
    App.PleaseWait.Show("Sending command to EP.");
    XmlNode cmdResult = App.SendCommandToEP(123);
    App.PleaseWait.Hide();
}
catch (Exception ex)
{
    //handle the error - see step 8
}
```

Command codes

- Create a **CommandCodes** class that defines the EdgeClick Processor command codes your application uses:

```
/// <summary>
/// Defines the EP command codes this app uses
/// </summary>
public static class CommandCodes
{
    public static int MY_CMD = 123;
}
```

Step 6 – Send Commands To EP

Command attributes

- To send additional values with a command, create an instance of **EdgeClick.Net.Http** with the attribute/value pairs added via the **AddTuple** method:

```
int someValue = 100;
EdgeClick.Net.Http attrs = new EdgeClick.Net.Http();
attrs.AddTuple("myAttribute", someValue.ToString());
```

- Then send the Http instance as an additional parameter:

```
XmlNode cmdResult =
     App.SendCommandToEP(CommandCodes.MY_CMD, attrs);
```

Step 7 – Parse Results

- The XML returned has the following format:

```
<return>
  <code>return_code</code>
  <result>XML_result</result>
</return>
```

- The **App.EPcodes** enum defines all the possible return codes.
- In practice the code will always be:
  - 0 – **App.EPcodes.SUCCESS** or,
  - 10 – **App.EPcodes.NEW_VERSION_AVAILABLE**.
- All other return codes will result in a specific exception being thrown.

Step 7 – Parse Results

- ```
  <return>
    <code>0</code>
    <result>
      <list>
        <System>
          <Name>acme.foo.com</Name>
          <OS>Windows XP</OS>
        </System>
        ...
      </list>
    </result>
  </return>
  ```

- ```
  string code = cmdResult.SelectSingleNode("/return/code").InnerText;
  XmlNode myResults = cmdResult.SelectSingleNode("/return/result");
  XmlNodeList systemList = myResults.FirstChild.ChildNodes;

  // parse each <System> node
  foreach (XmlNode system in systemList)
  {
     string systemName = system.SelectSingleNode("Name").InnerText;
     string systemOS = system.SelectSingleNode("OS").InnerText;
  }
  ```

Step 8 – Error Handing

- **EdgeClickNetworkException** – the EdgeClick Processor could not be reached, or the response was garbled.
- **EdgeClickApplicationException** – the command code, application index, version number or arguments were not valid, or the EdgeClick service itself threw an exception.
- **EdgeClickGenericException** – an unexpected error occurred.
- **EdgeClickUnsupportedVersionException** – this version of the client is unsupported, you should exit when you receive this exception.

Step 8 – Error Handing

- **EdgeClickAuthenticationException** – the current account has not been correctly authenticated, the user must authenticate an account before proceeding:

```
catch (EdgeClickAuthenticationException)
{
   App.PleaseWait.Hide();
   //get the user to create/specify an
   //authenticated account
   App.AccountMgr.SetCurrentAccount();
}
```

Step 8 – Error Handing

- **EdgeClickAuthExpiredException** – the session ID for the current account has expired, the user must enter their password to re-authenticate:

```
catch (EdgeClickAuthExpiredException)
{
    App.Pleasewait.Hide();

    //get the user to reauthenticate
    bool authenticated = false;
    authenticated = App.AccountMgr.ReAuthenticate();

    if (authenticated)
        //reauthenticated, resend the command
    else
        //authentication failed, disable everything
        //in the client except the EdgeClick menu
}
```

Step 9 – Version Control

- Eventually there will be support for automatic downloading and installing of new versions, but for now it must be done manually by the user.
- **SendCommandToEP** will throw an **EdgeClickUnsupportedVersionException** if the version number of the client is too old.
- If the version number of the client is older than the current version, but not old enough to be unsupported, the code section of **SendCommandToEP** will be set to **App.EPcodes.NEW_VERSION_AVAILABLE**.
- It is up to you how to handle this code, but it is recommended that it is checked for and a message displayed when sending the first command to theEdgeClick Processor, and ignored afterwards.

Step 9 – Version Control

```
code =
      cmdResult.SelectSingleNode("/return/code").InnerText;

if (code == App.EPcodes.NEW_VERSION_AVAILABLE)
{
   App.PleaseWait.Hide();
   MessageBox.Show("There is a new version available.",
                   "New Version",
                   MessageBoxButtons.OK,
                   MessageBoxIcon.Exclamation,
                   MessageBoxDefaultButton.Button1);
}
```

Packaging a .NET EdgeClick Client

- To install your application on a Windows Mobile smart device, you need three additional projects:
  - SmartDevice cab project that contains your application in a phone-installable .cab file.
  - custom installer project that will launch ActiveSync to install your application's cab file on the phone.
  - Setup project that creates a Windows installer to install the cab file onto a Windows desktop and then launch the custom installer to install it onto a connected Windows Mobile phone via ActiveSync.

SmartDeviceCab Project

- Create a new Smart Device CAB Project (Other Project Types, Setup and Deployment, Smart Device CAB project).
- Enter the appropriate values for properties like Manufacturer, ProductName etc.
- In the Properties Pages window, change the output filename from the default SmartDeviceCab.cab to something more useful.
- Select Add→Project output and add the "Primary output" from your main project.

SmartDeviceCab Project

- To include a shortcut to your application in the Programs folder on the phone:
  - Select View→File System, right-click on "File System on Target Machine" and select "Add special folder".
  - Choose "Programs Folder", right-click on it and then select "Create New Shortcut".
  - Browse to the application folder and select "Primary output" from your main project.

CustomInstaller Project

- Copy the CustomInstaller project from the Sample application included with the EdgeBuilder SDK to your solution's directory and add it to your solution.
- Open *CustomInstaller.cs*, find the reference to "Sample.ini" in the **CustomInstaller_AfterAction** method and change it to <your_app>.ini.
- Copy *Sample.ini* from the sample application to your main project, rename it to <your app>.ini and add it to your main project.
- Change the ini file's **Build Action** property to "Content" and **Copy to Output Directory** property to "Copy if newer".

CustomInstaller Project

- Edit the ini file, make changes apropos for your application:

```
[CEAppManager]
Version = 1.0
Component = Component

[Component]
Description = Your description
CabFiles    = <your_app>.cab
DeviceFile  = <your_app>.exe
```

- **Note:** the ini file format is very strict and it's easy to get the ActiveSync installer to fail with no obvious reason. Follow these rules:
  - **Version** must be in the format ***number.number***, eg. 1.0 – it cannot be a triplet of numbers.
  - **Component** must be a single word and match the section name in square brackets and the product name specified in the SmartDeviceCab project.

Windows Setup Project

- Create a new Setup project (Other Project Types, Setup and Deployment, Setup project).
- Enter appropriate values for properties like Manufacturer, ProductName, Version etc.
- Add the following project outputs to the Setup project:
  - "Built outputs" from the SmartDeviceCab project
  - "Primary output" from the CustomInstaller project
  - "Content files" from your main project
- Add "Primary output from CustomInstaller" as install and uninstall custom actions.

Handling different screen sizes

- Windows Mobile smart devices come in a variety of screen sizes, from 240x240 to 640x640.
- It is recommended that you design your UI with the smallest square size and allow your design to scale automatically to larger screens:
  - Create a panel that all other controls go inside and anchor it to all for sides of the screen.
  - If your UI has a list, or tab control, anchor them to all four sides of the screen.
  - If your UI design will not fit on the square screen size, switch to a larger size, create a panel that contains all the controls and set the AutoScroll property to true. Now when viewed on the smaller square screen, a scroll bar will appear, allowing the user to access all the controls.

Handling DateTimes

- Since an EdgeClick client can be run it disparate time zones from the EP or agent, handling DateTimes in a consistent manner is critical:
  - Always store and manipulate DateTimes in Coordinated Universal Time (UTC) – **myDateTime.ToUniversalTime()**
  - Display in local time – **myDateTime.ToLocalTime()**
  - Never use **DateTime.Now** for getting the current time, instead use **DateTime.UtcNow**
- These rules will ensure that a stored DateTime always refers to the exact same moment in time, no matter from where in the world it is viewed.

Q & A

- Questions?