

مقرر: برمجة التطبيقات الشبكية

الفصل الثاني: برمجة التطبيقات الشبكية زبون/مخدم

فهرس الفصل:

1. مقدمة.
2. مقارنة بين الاتصال المرتبط والاتصال عديم الارتباط.
3. بروتوكولات نقل البيانات.
4. بناء مخدم TCP باستخدام مقابس تدفق البيانات.
5. بناء زبون TCP باستخدام مقابس تدفق البيانات.
6. تفاعل زبون/مخدم باستخدام اتصالات مقابس تدفق البيانات.
7. تفاعل زبون/مخدم في الاتصال عديم الارتباط مع برقيات البيانات.
8. بناء تطبيق زبون/مخدم باستخدام مخدم متعدد النيايب.
9. عنصر التحكم.
10. التشبيك عن بعد.

1. مقدمة

يُمَنح التعامل مع شبكة الانترنت وعالم الويب الكثير من الإمكانيات نظراً لما تُقدِّمه من خدمات متنوعة، إضافةً إلى سهولة الحصول على المعلومات من أي مكان في العالم. وإذا فقد أدركت الشركات أهمية تشارك البيانات على شبكة الانترنت لتحسين الأداء.

تُقدم المكتبة البرمجية **NET FCL (Framework Class Library)** عدداً من الإجراءات الجاهزة الخاصة بالتشبيك والتي تُسهل عملية تطوير تطبيقات الويب وتطبيقات الانترنت. وبحيث تستطيع البرامج باستخدام الويب البحث عن المعلومات في كل مكان من العالم، كما تستطيع مشاركة برامج أخرى تعمل على حواسيب أخرى في أي منطقة بعيدة.

سنقوم في الفصل القادم بدراسة خدمات الويب **Web Services** المستخدمة في بناء التطبيقات الموزعة وذلك على المستوى العالي **high-level**.

أما في هذا الفصل، فسنبداً بعرض تقانات الاتصالات المستخدمة لإرسال البيانات عن طريق شبكة الانترنت، ثم نقوم بعرض الطرق الأساسية لتأسيس اتصال بين تطبيقين يستخدمان قناتين لتدفق البيانات بطريقة شبيهة لطريقة كتابة وقراءة الملفات **file I/O**. يدعى هذا الاتصال بالاتصال المرتبط **Connection Oriented** والذي يُمكِّن برنامجين من الاتصال ونقل البيانات بينهما بطريقة مشابهة للكتابة والقراءة من الملفات على الأقراص الصلبة. ثم نقوم بعرض تطبيق محادثة **Chatting** يستخدم هذه التقانات ويقوم بتبادل رسائل بين الزبون والمخدم.

نقوم بعد ذلك، بعرض مثال عن تقانة الاتصال عديم الارتباط **Connectionless** لنقل البيانات بين التطبيقات. والتي هي تقانة أقل وثوقية من الاتصال المرتبط إلا أنها أخف حملاً على الشبكة. تُستخدم هذه التقانة عادةً في تطبيقات نقل الصوت والفيديو عبر الانترنت.

ثم نعرض مثال على برنامج (زبون/مخدم) (لعبة بين لاعبين على الشبكة) يُبين من خلال طريقة بناء مخدم متعدد النواصب.

ندرس بعدها مكون التحكم الجديد **WebBrowser** والذي يسمح بإضافة آليات تصفح لأي تطبيق يستخدم الويب.

ينتهي هذا الفصل بعرض سريع عن تقانة التشبيك عن بعد **NET Remoting** والتي تُمكن خدمات مشابهة لخدمات الويب المدروسة في الفصل القادم والتي تسمح بالمعالجة الموزعة عبر الشبكة.

2. مقارنة بين الاتصال المرتبط والاتصال عديم الارتباط

Connection-Oriented vs. Connectionless Communication

يوجد طريقتين رئيسيتين لتحقيق الاتصال بين التطبيقات هما:

1. الاتصال المرتبط.

2. الاتصال عديم الارتباط.

يشابه الاتصال المرتبط النظام الهاتفي، حيث يتم تأسيس الاتصال وإبقائه مفتوحاً طالما هناك تبادل للمعلومات بين الطرفين. أي يبقي الاتصال مفتوحاً حتى إنهاء الاتصال من قبل أحده الطرفين.

بينما يشابه الاتصال عديم الارتباط الخدمة البريدية، حيث إذا قمنا بإرسال رسالتين من م-صدر ما إلى مكان ما آخر، فيمكن فعلياً أن تأخذ هاتين الرسالتين مسارين مختلفين في هذا النظام، كما يمكن أن تصلا في أوقات مختلفة، كما يمكن أن لاتصل إحدهما أبداً.

يتم في الاتصال المرتبط إرسال معلومات تحكم بين الحواسيب لتفعيل الاتصال بين الطرفين. تُسمى هذه التقنية بالمصافحة Handshaking.

تُعتبر شبكة الانترنت شبكة غير موثوقة، مما يعني أن المعلومات أو البيانات المرسلات عبر الانترنت يمكن أن تُفقد أو أن تتأذى خلال عملية النقل.

تُرسل البيانات عملياً في رزم Packets تحوي، إضافةً لمحتواها، مجموعة معلومات تساعد الانترنت في توجيه هذه الرزم لتصل إلى المكان المقصود والصحيح. لا تتضمن الانترنت لأحد ماذا سيحدث للرمز المرسل حيث يمكن أن تصل هذه الرزم معطوبة أو متآخرة أو فارغة أو مكررة كما يمكن أن لا تصل أبداً. يقتصر دور الانترنت على القيام بما في وسعها لإيصال الرزم المرسل.

يضمن الاتصال المرتبط تراسل معطيات موثوق ضمن شبكة غير موثوقة، ويقوم بتحقيق ضمان وصول الرزم المرسل إلى المستقبل صحيحة بدون أضرار، كما يفهم بتجميعها بالتسلسل الصحيح.

لا يقوم الحاسبان في الاتصال عديم الارتباط بالمصافحة قبل إرسال البيانات، ولا يضمن أحد وثوقية وصول هذه البيانات. يمكن للمعلومات أن لا تصل أبداً إلى المستقبل المقصود. وبهذا

النحو، يقوم الاتصال عديم الارتباط بتخفيف الحمل على الشبكة وذلك بتجنب إرسال معلومـات التحكم في تقنية المصافحة لضمان الوثوقية. وبذا فسنحتاج لإرسال معلومات أقل بين الحواسـب.

3. بروتوكولات نقل البيانات

يوجد العديد من البروتوكولات لتعريف كيفية الاتصال بين التطبيقات. يمكن تعريف البروتوكول بأنه مجموعة من القواعد التي تحكم كيفية التخاطب بين مكونين (يمكن أن تكون هذه المكونات عبارة عن حواسـب). سنقوم في هــذا الفـصل بـالتركيز عـلى بروتوكـول الـTCP Transmission Control Protocol و بروتوكـول الـUDP Datagram Protocol.

توفر NET إمكانيات التعامل مع البروتوكولات TCP و UDP من فضاء الأسـماء System.Net.Sockets.

يعتبر البروتوكول TCP بروتوكول مرتبط لأنه:

يضمن وصول رزم البيانات إلى الوجهة المقصودة بالتسلسل المرسل الصحيح وبدون أي فقدان أو ضياع في البيانات المرسلة.

يسمح البروتوكول TCP للبروتوكولات الأعلى مثل البروتوكول HTTP بإرسال المعلومات عبر شبكة الانترنت بطريقة مشابهة لكتابة الملفات على القرص الصلب للحاسب. يقوم البروتوكول TCP بإعادة إرسال رزم البيانات في حال ضياعها أو عدم وصولها إلى الوجهة المقصودة، كما يضمن تجميع هذه الرزم بالتسلسل الصحيح وبصورة واضحة للتطبيق المستلم، كما يفـيـقـوم البروتوكول باستبعاد الرزم المكررة.

يستخدم الاتصال عديم الارتباط UDP من قبل التطبيقات التي لا تتطلب بالضرورة تبادل معطيات موثوقة بين طرفي الاتصال. حيث يضمن البروتوكول UDP الحد الأدنى من نقل البيانات الضروري لتأمين الاتصال بين التطبيقات. لا يقدم البروتوكول UDP أي ضمان بأن رزم البيانات، والتي تسمى بـرقبات البيانات Datagrams، ستصل إلى الوجهة المقصودة أو بأنها ستصل بالترتيب التي أرسلت فيه.

يوجد العديد من الفوائد التي يمكن الحصول عليها عند استخدام البروتوكول UDP عوضاً عن البروتوكول TCP. فعند استخدام البروتوكول UDP تُشكّل عملية نقل البيانات بين طرفي

الاتصال حملاً أفل على الشبكة، إذ أن رزم البيانات لا تحتاج لأن تحمل معلومات التحكم التي يحتاجها البروتوكول TCP لضمان وثوقية وصول المعلومات.

كما أن البروتوكول UDP يقلل حركة البيانات على الشبكة نظراً لعدم اهتمامه بعملية المـصافحة وإعادة إرسال البيانات في حال عدم وصولها.

يمكن استخدام الاتصالات غير الموثوقة في العديد من الحالات العملية وحيث تكون الوثوقية غير مطلوبة وحيث نرغب بالتخلص من الحمل الزائد الذي تفرضه البروتوكولات التي تضمن وثوقية وصول البيانات.

فمثلاً يمكن لبعض التطبيقات التي تستخدم نقل الصوت والفيديو أن تتغاضى عن فقـدان بعض برقيات البيانات. يمكن بالطبع أن ينتج عن هذا الفقدان توقف مؤقت لعرض الصوت أو الفيديو. إلا أنه إذا كانت هذه التطبيقات تستخدم البروتوكول TCP فإن فقدان جزء من البيانات سـيؤدي إلى توقف كبير وملحوظ حيث أن التطبيق سينتظر حتى يتم إعادة إرسال الجزء المفقـود عند النقل واستقباله كي يتم مواصلة عمل التطبيق.

إضافةً إلى ذلك، فيمكن لبعض التطبيقات الحساسة والتي تحتاج لأن تعتمد تقنيات وثوقية خاصة مختلفة عن تلك التي يقدمها البروتوكول TCP أن تقوم بتطـوير هـذه التقنيات ابتداءً من البروتوكول UDP.

4. بناء مخدم TCP باستخدام مقابس تدفق البيانات Stream Sockets

يقوم المخدم عادةً بانتظار طلب الاتصال من الزبون. حيث يحتوي برنامج المخدم غالباً عبارات تحكم أو كتل برمجية تُنفذ باستمرار حتى يستقبل المخدم الطلب. عند وصول الطلب إلى المخدم يقوم بتأسيس الاتصال مع الزبون.

بعدها، يقوم المخدم باستخدام هذا الاتصال لمعالجة الطلبات المستقبلية للزبون وإرسال المعلومات إليه. تتم إدارة هذا الاتصال من خلال غرض من الصف Socket كما سنرى.

تقوم البرامج التي تتصل عن طريق البروتوكول TCP بمعالجة البيانات التي تستقبلها وترسلها كدفق من البايتات Bytes.

يتطلب تأسيس مخدم بسيط مع البروتوكول TCP ومقبس تدفق البيانات Stream Sockets خمسة مراحل:

1. المرحلة الأولى: إنشاء متنصت Listener.
2. المرحلة الثانية: بدء الاستماع لطلبات الاتصال.
3. المرحلة الثالثة: تأسيس الاتصال بين المخدم والزبون.
4. المرحلة الرابعة: معالجة الاتصال.
5. المرحلة الخامسة: إنهاء الاتصال.

المرحلة الأولى: إنشاء متنصت Listener

يجب أولاً إنشاء غرض من الصف TcpListener من فضاء الأسماء System.Net.Sockets. يمثل هذا الصف مقبس تدفق البيانات باستخدام TCP والذي يمكن للمخدم من خلاله الاستماع إلى الطلبات المرسله إليه.

يتم إنشاء منتسخ جديد من الصف TcpListener بالتعليمة التالية:

```
TcpListener server = new TcpListener( ipAddress , port )
```

تقوم التعليمة السابقة بربط تطبيق المخدم برقم منفذ محدد port. حيث يكون رقم المنفذ عبارة عن محدد رقمي يستخدم من قبل التطبيق لتعريف نفسه كمراقب لعنوان شبكة معطى.

يمكن أن تأخذ أرقام المنافذ قيم بين 0 و 65535. تقوم معظم أنظمة التشغيل بحجز أرقام المنافذ التي أرقامها أقل من 1024 وذلك من أجل خدمات النظام (مثل البريد الإلكتروني وخدمات الويب). ويجب أن تُمنح هذه التطبيقات مزايا خاصة للحصول على أرقام المنافذ المحجوزة.

يكون المعامل الأول للتعليمة السابقة ipAddress عن- وان IP Address Internet Protocol. تقوم عناوين IP بتعريف الحواسيب على شبكة الانترنت.

وفي الحقيقة، فإن أسماء المواقع مثل www.scs-org.sy هي مرادفات لعناوين IP. يمكن تعريف عناوين الشبكة من خلال الصف IPAddress من فضاء الأسماء System.Net.

يجب على أي تطبيق يقوم بعمليات على الشبكة أن يعرف نفسه عن طريق الزوج عن- وان IP- رقم منفذ. ولا يمكن لتطبيقين أن يأخذا رقم المنفذ ذاته لعنوان IP معطى.

سنرى أن ربط المقبس برقم منفذ (باستخدام الطريقة Bind من الصف Socket) هـ و غـ ر ضروري، لأن الصف TcpListener والصفوف الأخرى التي سنناقشها في هذا الفصل تصل تفـ وم بذلك بشكل آلي.

المرحلة الثانية: بدء الاستماع لطلبات الاتصال

يجب استدعاء الطريقة start من الصف TcpListener كـ يـيـ دأـ الغـ رض TcpListener بالاستماع لطلبات الإتصال.

المرحلة الثالثة: تأسيس الإتصال بين المخدم والزبون

يستمر المخدم بالاستماع لطلبات المرسله إليه بصورة مستمرة حتى يقوم أحد الزبائن بمحاولة الإتصال. بعدها ينشئ المخدم اتصال اعتماداً على الطلب المرسل.

يقوم غرض من الصف System.Net.Sockets.Socket بإدارة هذا الإتصال مع الزبون. تقوم الطريقة AcceptSocket من الصف TcpListener بقبول طلب الإتصال. تُعيد هــ هذه الطريقة غرض من النوع Socket نتيجة لإنشاء الإتصال:

```
Socket connection = server.AcceptSocket();
```

تقوم الطريقة AcceptSocket عند استقبال المخدم لطلب باستدعاء الطريقة Accept من الصف TcpListener لجعل الإتصال مفعلاً.

يخفي استخدام لغة C# التفاصيل المعقدة التي تتم على مستوى الشبكة عن المبرمج. حيث يمكن للمبرمج كتابة كود بسيط ومفهوم في برنامج المخدم والسماح للصفوف فـي فـضاء الأسماء System.Net.Socket بمعالجة تفاصيل قبول الطلبات وتأسيس الإتصال.

المرحلة الرابعة: معالجة الاتصال

يتم في هذه المرحلة استقبال وإرسال الطلبات والمعلومات بين المخدم والزبون عن طريق الطريقتين Send و Receive من الصف Socket. يجب ملاحظة أن هاتين الطريقتين يمكن استخدامهما فقط في حالة تحقيق الإتصال فعلياً بين المخدم والزبون. بينما يمكن استخدام مقاييس برقية البيانات والبروتوكول UDP من خلال التوايع SendTo و RecieveFrom من الصف Socket في حال عدم وجود اتصال كما سنرى لاحقاً.

المرحلة الخامسة: إنهاء الاتصال

عند انتهاء المخدم والزبون من عملية استقبال وإرسال الطلبات والمعلومات، يستخدم المخدم الطريقة Close من الصف Socket لإنهاء الإتصال. بعدها يعود المخدم إلى المرحلة الثانية (انتظار طلب اتصال آخر من قبل الزبون).

يوجد مشكلة في المخطط السابق لتأسيس الاتصال بين المخدم والزيون ذلك أنه عند معالجة طلب من قبل المخدم في المرحلة الرابعة، تقوم هذه المعالجة بحجب الطلبات الأخرى من أن تُخدَم. وبهذا، فلا يمكن لأي زيون أن يتصل مع المخدم ما دام المخدم منهمكاً في معالجة طلبات الزيون الأول المتصل. ولحل هذه المشكلة، يجب استخدام البرمجة متعددة النياسـ ب **Multi Threading** والتي درسناها في الفصل السابق. فعند استقبال طلب اتصال من قبل المخدم، يقوم الأخير بإنشاء نيسب **Thread** جديد لتخديم الطلب الجديد مما يسمح للمفـس أو **TcpListener** مـ ن اسـتقبال اتصالات أخرى.

المرحلة الخامسة: إنهاء الاتصال

لاحظ أن استخدام الإمكانيات التي تقدمها تقنية تعددية النياسب في **C#** تجعل في الإمكان بناء خدمات تقوم بإدارة اتصالات متزامنة مع أكثر من زيون. تُستخدم منهجية تعددية النياسب بشكل كبير في خدمات الشبكة لأنظمة التشغيل **Windows** و **UNIX**.

يمكن أن يزود المخدم متعدد النياسب في الأنظمة ذات الأداء العالي والذواكر الـ وفيرة بألفة على بناء بركة من النياسب **Pool**. يمكن لهذه النياسب معالجة عمليات الدخـل/الخـرج على الشبكة بصورة سريعة وفعالة عبر مقابس متعددة، وبحيث يستطيع المخدم معالجة الاتصالات الـ واردة إليه دون الحاجة لبناء نيسب جديد عند كل اتصال مما يساعد على زيادة الأداء.

5. بناء زيون TCP باستخدام مقابس تدفق البيانات **Stream Sockets**

يتطلب بناء تطبيق زيون يعتمد على مقابس تدفق البيانات **Stream Sockets** أربعة مراحل:

1. المرحلة الأولى: إنشاء زيون TCP

يتم في هذه المرحلة بناء غرض من الصف **TcpClient** (والذي ينتمي إلـى فـضاء الأسـماء **System.Net.Socket**) للاتصال بالمخدم.

يؤسس هذا الاتصال باستخدام الطريقة **Connect** من الصف **TcpClient**. والتي تملك معـاملتي دـخـل: الأول هو عنوان المخدم **IP**، والثاني هو رقم المنفذ.

```
TcpClient client = new TcpClient ();  
client.Connect( serverAddress, serverPort );
```

حيث يكون المعامل `serverPort` رقم منفذ، أما المعامل `serverAddress` فيمكن أن يكون منتسج من الصف `IPAddress` (الذي يغلف عنوان IP للمخدم) أو سلسلة حرفية تُعرف باسم المخدم.

يمكن أيضاً استدعاء الطريقة `Connect` مع معامل دخل وحيد من النمط مرجع لغرض من الصف `IPEndPoint` والذي يمثل الزوج عنوان-منفذ لنسخة أخرى من التحميل الزائد للطريقة `Connect`.

تستدعي الطريقة `Connect` من الصف `TcpClient` الطريقة `Connect` من الصف `Socket` لتأسيس الاتصال. إذا نجحت عملية الاتصال تقوم الطريقة `TcpClient.Connect` بإعادة رقم موجب وإلا فتُعيد 0.

2. المرحلة الثانية: توفير تدفق معطيات شبكي

يستخدم الصف `TcpClient` في هذه المرحلة الطريقة `GetStream` للحصول على تدفق معطيات شبكي `NetworkStream` والذي يسمح له بالقراءة من والكتابة إلى المخدم.

يمكن استخدام الطريقتين `WriteByte` و `Write` لكتابة بايت وحيد أو سلسلة من البايتات على منفذ تدفق البيانات على الشبكة لتصل إلى المخدم. كما يمكن استخدام الطريقتين `ReadByte` و `Read` لقراءة بايت أو سلسلة من البايتات من كالمخدم.

3. المرحلة الثالثة: معالجة الاتصال

يتم في هذه المرحلة استقبال وإرسال الطلبات والمعلومات بين المخدم والزيون. يستخدم الزيون في هذه المرحلة الطرائق `Read` و `ReadByte` و `Write` و `WriteByte` من الصف `NetworkStream` لإنجاز العمليات المطلوبة خلال الاتصال.

يمكن للزيون أيضاً استخدام النيايب لمنع عملية حجز الاتصال مع المخدمات الأخرى بينما يتم تخديم ومعالجة أحد الاتصالات.

4. المرحلة الرابعة: إغلاق الاتصال

بعد إنهاء عملية نقل البيانات، تتطلب المرحلة الرابعة من الزيون إغلاق الاتصال باستخدام الطريقة `Close` من الغرض `NetworkStream` والتي تقوم بإغلاق المقابس الموافقة لعملية الاتصال (إذا كان الغرض `NetworkStream` يحوي مرجعاً لذلك المقبس). بعدها يقوم الزيون باستدعاء الطريقة `Close` من الصف `TcpClient` لإنهاء الاتصال `TCP`. يمكن بعد ذلك طلب وتأسيس اتصال جديد باستدعاء الطريقة `Connect` من جديد.

6. تفاعل زبون/مخدم باستخدام اتصالات مقابس تدفق البيانات

تستخدم التطبيقات التالية الصفوف والتقانات المناقشة في الفقرتين السابقتين لبناء تطبيق محادثة زبون/مخدم بسيط. حيث ينتظر المخدم طلبات الزبون لتحقيق اتصال. وعندما يتصل تطبيق الزبون بالمخدم، يرسل تطبيق المخدم سلسلة من البايتات إلى الزبون تشير إلى نجاح عملية الاتصال. يعرض عندها الزبون رسالة لإعلام المستخدم بنجاح هذه العملية وإتمام عملية الاتصال.

يحتوي، في المثال التالي، كل من تطبيق المخدم والزبون على مربع نص يمنح المستخدم إمكانية كتابة رسالة وإرسالها إلى التطبيق الآخر. إذا أرسل تطبيق المخدم أو تطبيق الزبون الرسالة TERMINATE فإن الاتصال بين المخدم والزبون ينتهي.

المخدم

يكون في هذا التطبيق كود المخدم:

```
1 // ChatServer.cs
2 // Set up a server that will receive a connection from a client
3 // send a string to client, chat with the client and close
4 using System;
5 using System.Windows.Forms;
6 using System.Threading;
7 using System.Net;
8 using System.Net.Sockets;
9 using System.IO;
10
11 public partial class ChatServerForm : Form
12 {
13     public ChatServerForm()
14     {
15         InitializeComponent();
16     } // end constructor
17
18     private Socket connection; // Socket for accepting a connection
19     private Thread readThread; // Thread for processing incoming msgs
20     private NetworkStream socketStream; // network data stream
21     private BinaryWriter writer; // facilitates writing to the stream
22     private BinaryReader reader; // facilitates reading from the stream
23
24     // initialize thread for reading
25     private void ChatServerForm_Load( object sender, EventArgs e )
26     {
27         readThread = new Thread( new ThreadStart( RunServer ) );
28         readThread.Start();
29     } // end method ChatServerForm_Load
30
31     // close all threads associated with this application
32     private void ChatServerForm_FormClosing( object sender,
```

```

33     FormClosingEventArgs e )
34     {
35         System.Environment.Exit( System.Environment.ExitCode );
36     } // end method CharServerForm_FormClosing
37
38     // delegate that allows method DisplayMessage to be called
39     // in the thread that creates and maintains the GUI
40     private delegate void DisplayDelegate( string message );
41
42     // method DisplayMessage sets displayTextBox's Text property
43     // in a thread-safe manner
44     private void DisplayMessage( string message )
45     {
46         // if modifying displayTextBox is not thread safe
47         if ( displayTextBox.InvokeRequired )
48         {
49             // use inherited method Invoke to execute DisplayMessage
50             // via a delegate
51             Invoke( new DisplayDelegate( DisplayMessage ),
52                 new object[] { message } );
53         } // end if
54         else // OK to modify displayTextBox in current thread
55             displayTextBox.Text += message;
56     } // end method DisplayMessage
57
58     // delegate that allows method DisableInput to be called
59     // in the thread that creates and maintains the GUI
60     private delegate void DisableInputDelegate( bool value );
61
62     // method DisableInput sets inputTextBox's ReadOnly property
63     // in a thread-safe manner
64     private void DisableInput( bool value )
65     {
66         // if modifying inputTextBox is not thread safe
67         if ( inputTextBox.InvokeRequired )
68         {
69             // use inherited method Invoke to execute DisableInput
70             // via a delegate
71             Invoke( new DisableInputDelegate( DisableInput ),
72                 new object[] { value } );
73         } // end if
74         else // OK to modify inputTextBox in current thread
75             inputTextBox.ReadOnly = value;
76     } // end method DisableInput
77
78     // send the text typed at the server to the client
79     private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
80     {
81         // send the text to the client
82         try
83         {
84             if ( e.KeyCode == Keys.Enter && inputTextBox.ReadOnly==false)
85             {
86                 writer.Write( "SERVER>>> " + inputTextBox.Text );
87                 displayTextBox.Text += "\r\nSERVER>>> " + inputTextBox.Text;
88
89                 // if the user at the server signaled termination
90                 // sever the connection to the client
91                 if ( inputTextBox.Text == "TERMINATE" )

```

```

92         connection.Close();
93
94         inputTextBox.Clear(); // clear the user's input
95     } // end if
96 } // end try
97 catch ( SocketException )
98 {
99     displayTextBox.Text += "\nError writing object";
100 } // end catch
101 } // end method inputTextBox_KeyDown
102
103 // allows a client to connect; displays text the client sends
104 public void RunServer()
105 {
106     TcpListener listener;
107     int counter = 1;
108
109     // wait for a client connection and display the text
110     // that the client sends
111     try
112     {
113         // Step 1: create TcpListener
114         IPAddress local = IPAddress.Parse( "127.0.0.1" );
115         listener = new TcpListener( local, 50000 );
116
117         // Step 2: TcpListener waits for connection request
118         listener.Start();
119
120         // Step 3: establish connection upon client request
121         while ( true )
122         {
123             DisplayMessage( "Waiting for connection\r\n" );
124
125             // accept an incoming connection
126             connection = listener.AcceptSocket();
127
128             // create NetworkStream object associated with socket
129             socketStream = new NetworkStream( connection );
130
131             // create objects for transferring data across stream
132             writer = new BinaryWriter( socketStream );
133             reader = new BinaryReader( socketStream );
134
135             DisplayMessage("Connection "+counter+" received.\r\n" );
136
137             // inform client that connection was successful
138             writer.Write( "SERVER>>> Connection successful" );
139
140             DisableInput( false ); // enable inputTextBox
141
142             string theReply = "";
143
144             // Step 4: read string data sent from client
145             do
146             {
147                 try
148                 {
149                     // read the string sent to the server
150                     theReply = reader.ReadString();

```

```

151
152         // display the message
153         DisplayMessage( "\r\n" + theReply );
154     } // end try
155     catch ( Exception )
156     {
157         // handle exception if error reading data
158         break;
159     } // end catch
160 } while ( theReply != "CLIENT>>> TERMINATE" &&
161         connection.Connected );
162
163 DisplayMessage( "\r\nUser terminated connection\r\n" );
164
165 // Step 5: close connection
166 writer.Close();
167 reader.Close();
168 socketStream.Close();
169 connection.Close();
170
171 DisableInput( true ); // disable InputTextBox
172 counter++;
173 } // end while
174 } // end try
175 catch ( Exception error )
176 {
177     MessageBox.Show( error.ToString() );
178 } // end catch
179 } // end method RunServer
180 } // end class ChatServerForm

```

قبول الاتصال وتأسيس مجرى تدفق البيانات

تُعرف الأسطر 121-173 حلقة لا نهائية تبدأ بتأسيس الاتصال المطلوب من قبل الزبون (المرحلة 3). يستدعى السطر 126 الطريقة `AcceptSocket` للغرض `TcpListener` والذي يعيد `Socket` نتيجة للاتصال الناجح.

يتم حجز النيسب الذي تستدعيه الطريقة `AcceptSocket` كي يتم تأسيس الاتصال. يدير الغرض المعاد من نمط `Socket` عملية الاتصال.

يقوم السطر 129 بتمرير الغرض `Socket` كمعامل لباني الغرض `NetworkStream` والذي يزود إمكانية النفاذ الى قنوات تدفق البيانات عبر الشبكة.

يستخدم الغرض `NetworkStream` في هذا المثال قنوات تدفق البيانات لمقبس محدد.

تتم -وم الأسطر 132-133 بيناء متتسحات instances من الـ صنفوف `BinaryWriter` و `BinaryReader` من أجل كتابة وقراءة البيانات، حيث يتم تمرير الغرض `NetworkStream` كمعامل عند بناء هذه المنتسحات.

يمكن للكائن `BinaryWriter` كتابة بايتات على مجرى التدفق الشبكي `NetworkStream` و `BinaryReader` كما يمكنه القراءة منها.

يقوم السطر 135 باستدعاء الطريقة `DisplayMessage` مشيراً إلى وصول طلب الإتصال إلى المخدم، بعدها يتم إرسال رسالة للزبون لإعلامه بوصول طلب الإتصال.

يوجد للطريقة `Write` الخاصة بالصف `BinaryWriter` عدة نسخ من التحميل الزائد (`overload versions`) يمكنها كتابة البيانات على مجرى تدفق البيانات بطرق مختلفة.

يستخدم السطر 138 الطريقة `Write` لإرسال سلسلة محرفية للزبون تقوم بإعلامه بنجاح عملية الإتصال. وبهذا تكون انتهت المرحلة الثالثة.

استقبال رسائل من الزبون

بعد قبول الاتصال تبدأ مرحلة المعالجة (المرحلة 4). تُعرف الأسطر 145-161 حلقة `do... while` يتم تنفيذها بصورة مكررة إلى أن يستقبل المخدم رسالة تشير إلى إنهاء الاتصال (هذا `(CLIENT>>> TERMINATE)`).

يستخدم السطر 150 الطريقة `ReadString` من الصف `BinaryReader` لقراءة سلسلة محرفية من مجرى تدفق الشبكة. يتم تجميد الطريقة `ReadString` حتى تتم عملية القراءة، وهذا هو سبب تنفيذ الطريقة `RunServer` ضمن نيسب مستقل (يتم بناء هذا النيسب في الأسطر 27-28 عند تحميل النموذج). يضمن هذا النيسب لمستخدم تطبيق المخدم بالتفاعل مع واجهة التطبيق وإرسال رسائل إلى الزبون حتى في حال تجميد النيسب المسؤول عن عملية القراءة وهو في حالة انتظار رسالة من الزبون.

تعديل عناصر التحكم في واجهة التطبيق من خلال نيسب منفصلة

بينما في الفصل السابق بأن عناصر التحكم في النماذج `Forms` غير آمنة للنيسب `Thread` (safe)، أي أنه عند تعديل عنصر التحكم من قبل أكثر من نيسب فلا يمكن ضمان إيجراءه. هذا التعديل بصورة صحيحة.

كما عرضنا في الفصل السابق، وجوب استخدام الطريقة `Invoke` للصف `Control` لتحفيق سلامة التعديل. تملك الطريقة `Invoke` معاملي دخل: المعامل الأول هو عبارة عن مفعول `delegate` يمثل الطريقة التي سيتم من خلالها تعديل واجهة التطبيق، والثاني مصفوفة من الأغراض تُمثل معاملات هذه الطريقة.

يعرف السطر 40 المفوض DisplayDelegate الذي يمثل طريقة تأخذ سلسلة محرفية كمعامل. ل دخل ولا تُعيد أي قيمة.

توافق الطريقة DisplayMessage (الأسطر 44-56) المفوض السابق حيث تستقبل سلسلة محرفية كمعامل باسم message ولا تُعيد أي قيمة.

تقوم التعليمة الشرطية if في السطر 47 بفحص الخاصية InvokeRequired لمعرفة المدع المعطى. DisplayTextBox (يتم وراثته هذه الخاصية من الصف Control) والتي تعيد القيمة True في حال كان من غير المسموح للنيسب الحالي التعديل على عنصر التحكم مباشرة وإلا يعيد القيمة False.

إذا كان النيسب الحالي والذي يقوم بتنفيذ الطريقة DisplayMessage مغاير للنيسب الذي قام ببناء واجهة التطبيق، فعندها تُعطي التعليمة if القيمة true، وتستدعي الأسطر 51-52 الطريقة invoke ممررةً لها DisplayDelegate الذي يمثل الطريقة DisplayMessage نفسها كمعامل. أول ومصفوفة جديدة من الأغراض تتكون من سلسلة محرفية كمعامل ثاني.

مما يجعل النيسب الذي قام ببناء واجهة التطبيق يقوم باستدعاء الطريقة DisplayMessage ثانية في وقت لاحق مع نفس المعامل للاستدعاء الأصلي. عندما يحدث ذلك الاستدعاء من النيسب الذي قام ببناء واجهة التطبيق، فإن الطريقة قادرة حينها على تعديل DisplayTextBox مباشرة ويتم تنفيذ الترميز ضمن عبارة else (السطر 55) وإضافة رسالة إلى المدع الخاص بـ DisplayTextBox.

تقوم الأسطر 60-76 بتعريف المفوض DisableInputDelegate والطريقة DisableInput والتي تسمح للنيسب بتعديل الخاصية ReadOnly للعنصر DisplayTextBox باستخدام نفس التقنية السابقة. يستدعي النيسب الطريقة DisableInput مع تمرير معامل من نوع bool (True مقابل لفعال و False مقابل لغير فعال).

إذا كان من غير المسموح للطريقة DisableInput بتعديل عناصر التحكم أثناء تنفيذ النيسب الحالي، فعندها تقوم الطريقة DisableInput باستدعاء الطريقة Invoke. مما يسمح للنيسب الذي قام ببناء واجهة التطبيق بتعديل قيمة الخاصية inputTextBox.ReadOnly حسب القيمة الممررة في وقت لاحق.

إنهاء الإتصال مع الزبون

عند انتهاء عملية المحادثة. يتم في الأسطر 166-169 إغلاق كـل مـن BinaryWriter و BinaryReader و NetworkStream و Socket عن طريق استدعاء الطريقة Close لكـل منهم. بعدها يقوم المخدم بانتظار طلب اتصال آخر عن طريق العودة لتنفيذ حلقة while مـن البداية.

إرسال رسالة إلى الزبون

عند إدخال مستخدم تطبيق المخدم سلسلة محرفية في مربع النص inputTextBox والقيام بضغط المفتاح Enter، يقوم معالج الحدث inputTextBox_KeyDown (الأسطر 79-101) بـقراءة السلسلة المحرفية وإرسالها باستخدام الطريقة Write مـن الـصف BinaryWriter. إذا أنهى المستخدم تطبيق المخدم، يقوم السطر 92 باستدعاء الطريقة Close للـغـض Socket لإنهاء الاتصال.

إنهاء تطبيق المخدم

تُعرف الأسطر 32-36 معالج الحدث ChatServerForm_FormClosing من أجل تنفيذه عند إغلاق التطبيق. يقوم هذا الحدث بإغلاق التطبيق عن طريق استدعاء الطريقة Exit من الـصف Environment مع تمرير المعامل ExitCode لإنهاء كل النيات الخاصة بهذا التطبيق.

الزبون

يكون في هذا التطبيق كود الزبون:

```
1 // ChatClient.cs
2 // Set up a client that will send information to and
3 // read information from a server.
4 using System;
5 using System.Windows.Forms;
6 using System.Threading;
7 using System.Net.Sockets;
8 using System.IO;
9
10 public partial class ChatClientForm : Form
11 {
12     public ChatClientForm()
13     {
14         InitializeComponent();
15     } // end constructor
16
17     private NetworkStream output;//stream for receiving data
```

```

18 private BinaryWriter writer; //facilitates writing to the stream
19 private BinaryReader reader; //facilitates reading from the stream
20 private Thread readThread; //Thread for incoming messages
21 private string message = "";
22
23 // initialize thread for reading
24 private void ChatClientForm_Load( object sender, EventArgs e )
25 {
26     readThread = new Thread( new ThreadStart( RunClient ) );
27     readThread.Start();
28 } // end method ChatClientForm_Load
29
30 // close all threads associated with this application
31 private void ChatClientForm_FormClosing( object sender,
32     FormClosingEventArgs e
33 {
34     System.Environment.Exit( System.Environment.ExitCode );
35 } // end method ChatClientForm_FormClosing
36
37 // delegate that allows method DisplayMessage to be called
38 // in the thread that creates and maintains the GUI
39 private delegate void DisplayDelegate( string message );
40
41 // method DisplayMessage sets displayTextBox's Text property
42 // in a thread-safe manner
43 private void DisplayMessage( string message )
44 {
45     // if modifying displayTextBox is not thread safe
46     if ( displayTextBox.InvokeRequired )
47     {
48         // use inherited method Invoke to execute DisplayMessage
49         // via a delegate
50         Invoke( new DisplayDelegate( DisplayMessage ),
51             new object[] { message } );
52     } // end if
53     else // OK to modify displayTextBox in current thread
54         displayTextBox.Text += message;
55 } // end method DisplayMessage
56
57 // delegate that allows method DisableInput to be called
58 // in the thread that creates and maintains the GUI
59 private delegate void DisableInputDelegate( bool value );
60
61 // method DisableInput sets inputTextBox's ReadOnly property
62 // in a thread-safe manner
63 private void DisableInput( bool value )
64 {
65     // if modifying inputTextBox is not thread safe
66     if ( inputTextBox.InvokeRequired )
67     {
68         // use inherited method Invoke to execute DisableInput
69         // via a delegate
70         Invoke( new DisableInputDelegate( DisableInput ),
71             new object[] { value } );
72     } // end if
73     else // OK to modify inputTextBox in current thread
74         inputTextBox.ReadOnly = value;
75 } // end method DisableInput
76

```

```

77 // sends text the user typed to server
78 private void inputTextBox_KeyDown(object sender, KeyEventArgs e)
79 {
80     try
81     {
82         if (e.KeyCode==Keys.Enter && inputTextBox.ReadOnly==false)
83         {
84             writer.Write( "CLIENT>>> " + inputTextBox.Text );
85             displayTextBox.Text += "\r\nCLIENT>>> "+inputTextBox.Text;
86             inputTextBox.Clear();
87         } // end if
88     } // end try
89     catch ( SocketException )
90     {
91         displayTextBox.Text += "\nError writing object";
92     } // end catch
93 } // end method inputTextBox_KeyDown
94
95 // connect to server and display server-generated text
96 public void RunClient()
97 {
98     TcpClient client;
99
100     // instantiate TcpClient for sending data to server
101     try
102     {
103         DisplayMessage( "Attempting connection\r\n" );
104
105         // Step 1: create TcpClient and connect to server
106         client = new TcpClient();
107         client.Connect( "127.0.0.1", 50000 );
108
109         // Step 2: get NetworkStream associated with TcpClient
110         output = client.GetStream();
111
112         // create objects for writing and reading across stream
113         writer = new BinaryWriter( output );
114         reader = new BinaryReader( output );
115
116         DisplayMessage( "\r\nGot I/O streams\r\n" );
117         DisableInput( false ); // enable inputTextBox
118
119         // loop until server signals termination
120         do
121         {
122             // Step 3: processing phase
123             try
124             {
125                 // read message from server
126                 message = reader.ReadString();
127                 DisplayMessage( "\r\n" + message );
128             } // end try
129             catch ( Exception )
130             {
131                 // handle exception if error in reading server data
132                 System.Environment.Exit(System.Environment.ExitCode);
133             } // end catch
134         } while ( message != "SERVER>>> TERMINATE" );
135

```

```

136 // Step 4: close connection
137 writer.Close();
138 reader.Close();
139 output.Close();
140 client.Close();
141
142 Application.Exit();
143 } // end try
144 catch ( Exception error )
145 {
146 // handle exception if error in establishing connection
147 MessageBox.Show( error.ToString(), "Connection Error",
148     MessageBoxButtons.OK, MessageBoxIcon.Error );
149 System.Environment.Exit( System.Environment.ExitCode );
150 } // end catch
151 } // end method RunClient
152 } // end class ChatClientForm

```

يقوم ChatClientForm ببناء نيسب (الأسطر 96-151) لمعالجة الرسائل الواردة.

تتصل الطريقة RunClient بالصف ChatServerForm حيث تقـوم بالـستقبال البيانات منـه وإرسال البيانات إليه.

تقوم الأسطر 106-107 بتعريف وتهئية الغرض TcpClient ثم استدعاء الطريقة Connect لتأسيس عملية الإتصال مع المخدم (المرحلة 1). يكون المعامل الأول للطريقة Connect هــ واسم المخدم (في حالتنا هذه اسم المخدم هو localhost مما يعني أن المخدم والزيون يتوضـعان على نفس الحاسب). يطابق localhost لعنوان IP للحاسب المـضيف وهــ 127.0.0.1. أمـا المعامل الثاني فهو رقم المنفذ والذي ينتظر المخدم من خلاله لاستقبال طلبات الاتصال.

يستخدم ChatClientForm غرض من الصف NetworkStream لاستقبال وإرسال البيانات إلى المخدم.

يتضمن الزيون في السطر 110 على غرض من الصف NetworkStream نتيجة لـ استدعاء الطريقة GetStream من الصف TcpClient (المرحلة 2). يستمر تنفيذ الحلقة do...while فـي الأسطر 120-134 بصورة متكررة حتى يتم استقبال رسالة TERMINATE من قبل المخدم.

يستخدم السطر 126 الطريقة ReadString من الصف BinaryReader للحصول على الرسالة التالية من المخدم (المرحلة 3). يعرض السطر 127 هذه الرسالة، ثم تقـوم الأسـطر 137-140 بإغلاق الاتصال لكل من الإغراض BinaryWriter و BinaryReader و NetworkStream و TcpClient (المرحلة 4).

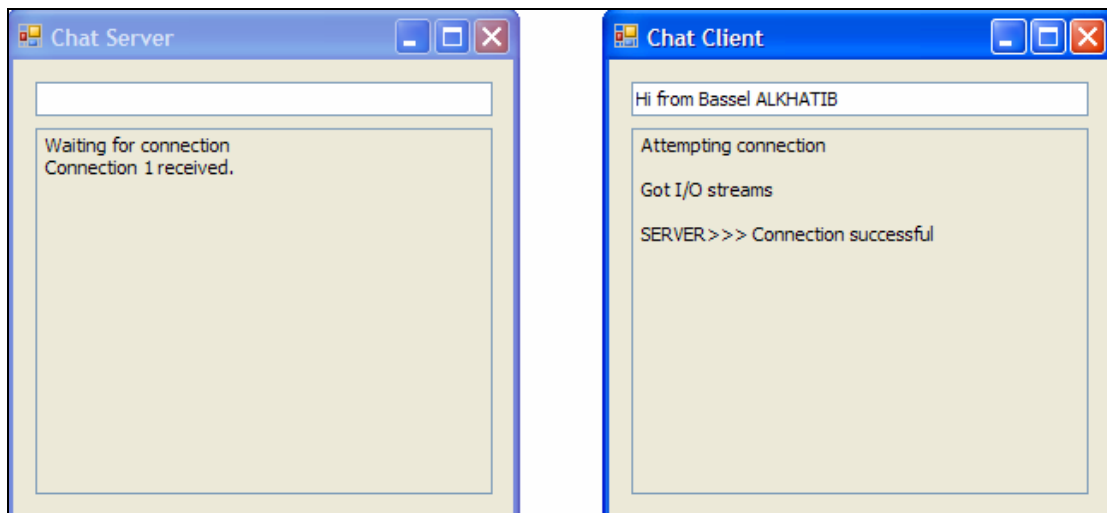
تُعدّ -- رف الأسس -- طر 75-39 كـ -- ل م -- ن DisplayMessage و DisplayDelegate و DisableInput و DisableInputDelegate بصورة مشابهة تماماً لما هو معـ رف سـ ابقاً فـ في الأسطر 76-40 من كود المخدم. تُستخدم هذه الطرائق لضمان تعديل واجهة التطبيق من النيسب الذي قام ببناء واجهة التطبيق فقط.

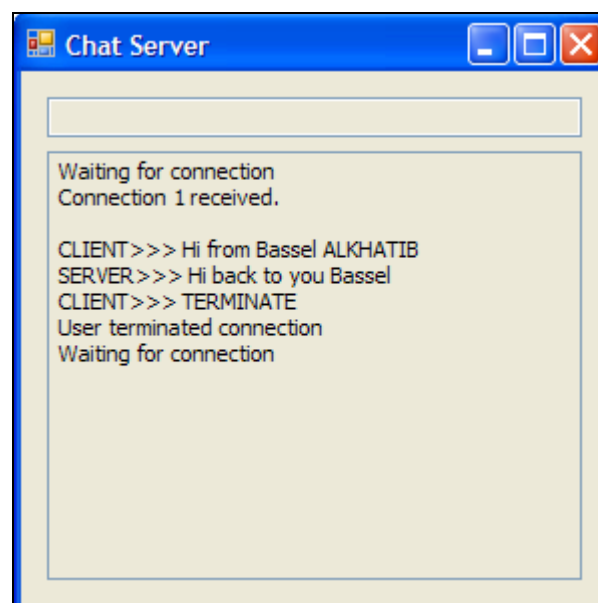
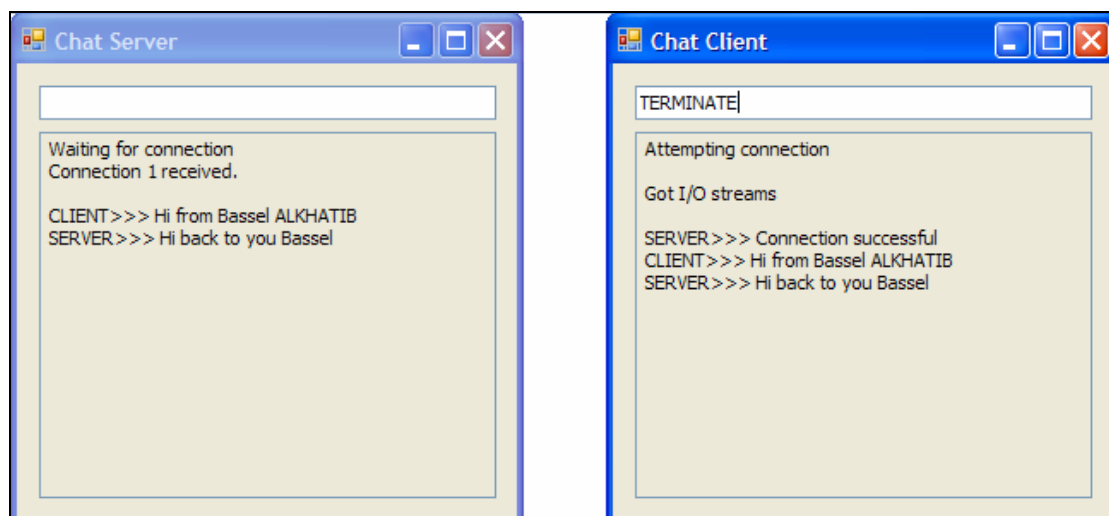
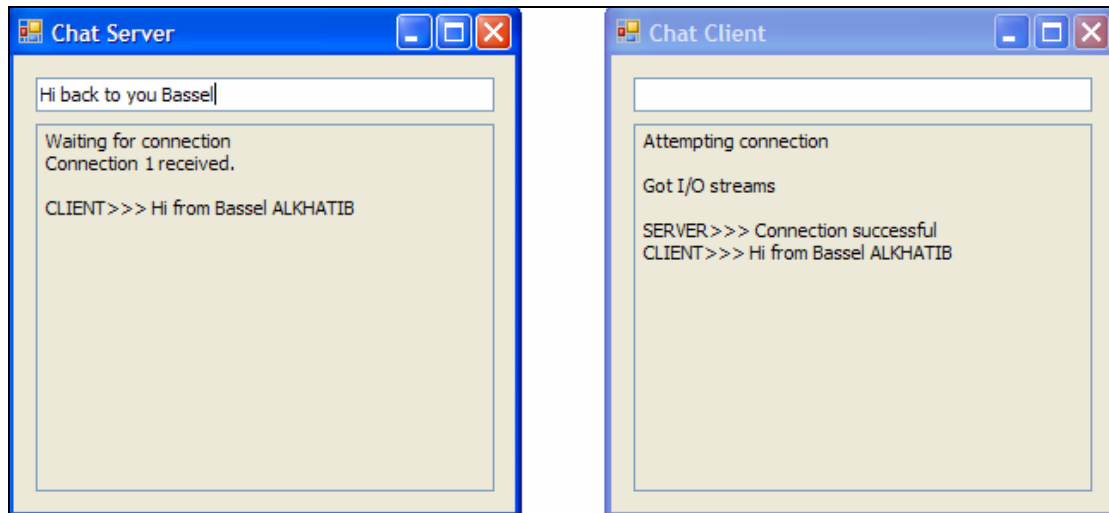
عندما يقوم مستخدم تطبيق الزبون بإدخال سلسلة محرفية في مربع النص ومن ثم ضغط المفتاح Enter، يقوم معالج الحدث `inputTextBox_KeyDown` بقراءة السلسلة المحرفية وإرسالها باستخدام الطريقة `Write` من الصف `BinaryWriter`.

نلاحظ هنا أن:

`ChatServerForm` يستقبل الاتصال ثم يقوم بمعالجته، بعدها يتم إغلاق الاتصال لينتظر من جديد من أجل طلب اتصال جديد. لكن إذا كنا نتحدث عن تطبيق واقعي فيقوم المخدم باستقبال الاتصال ثم إسناد معالجته لنيسب منفصل ليقيم بتنفيذ عميلة الاتصال، ليتابع انتظار طلب اتصال جديد. تقوم النيسب المنفصلة بمعالجة الاتصالات الحالية بينما يقوم المخدم بالتركيز على طلبات الاتصال الجديدة.

تُبين الأشكال التالية مختلف مراحل تفاعل المخدم والزبون:





7. تفاعل زبون/مخدم في الاتصال عديم الارتباط مع برفيات البيانات

Connectionless Client/Server Interaction With Datagrams

عرضنا في الفقرة السابقة الاتصال المرتبط ونقل البيانات اعتماداً على تدفق البيانات Streams-Based باستخدام البروتوكول TCP وذلك لضمان وصول رزم البيانات إلى الوجهة المقصودة بصورة موثوقة.

نعتبر فيما يلي أن إرسال البيانات يتم باستخدام الاتصال عديم الارتباط أي باستخدام برفية البيانات Datagram والبروتوكول UDP.

تُشابه عملية إرسال البيانات باستخدام الاتصال عديم الارتباط واستخدام برفيات البيانات عملية تسليم الرسائل في خدمة البريد.

تتم معالجة البيانات وإرسالها عند استخدام الاتصال عديم الارتباط كرمز معطيات تُدعى برفيات البيانات.

يمكن تشبيه برفية البيانات بالرسائل المرسلة عن طريق البريد. في حال عدم القدرة على إرسال رسالة كبيرة ضمن مغلف واحد، يتم تقسيم هذه الرسالة إلى أقسام، ثم يتم إرسالها ضمن مغلفات منفصلة وبصورة متسلسلة.

كما تُرسل كل هذه الرسائل مرة واحدة. يمكن أن تصل هذه الرسائل بصورة مرتبة أو غير مرتبة أو أن لا تصل أبداً. عند استقبال الرسائل من قبل الشخص عند نقطة الاستقبال يتم بتجميع الأجزاء المرسلة بالترتيب الصحيح قبل أن يحاول قراءة هذه الرسالة. إذا كانت الرسالة صغيرة كفاية بحيث يمكن إرسالها ضمن مغلف واحد، فعندها لا يبيّن أي مشكلة في تسليم الإرسال، لكن تبقى إمكانية عدم وصول الرسالة أبداً بعد عملية الإرسال.

تُوفّر C# الصف UdpClient من أجل إجراء نقل البيانات في الاتصال عديم الارتباط. يستخدم الصف UdpClient طرائق من الصف Socket بطريقة مشابهة لاستخدام الصف Socket و TcpListener و TcpClient لطرائق من الصف Socket كما رأينا سابقاً. تقوم الطرائق Send و Receive الخاصة بالصف UdpClient باستخدام الطرائق SendTo و ReceiveForm من الصف Socket لإرسال واستقبال البيانات.

تستخدم البرامج المعروضة فيما يلي برفية البيانات لإرسال رزم من البيانات بـين تطبيقات الزبون والمخدم. يقوم المستخدم في التطبيق PacketClient بكتابة رسالة في مربع النص TextBox ثم يقوم بضغط المفتاح Enter.

يقوم الزبون بتحويل هذه الرسالة إلى سلسلة من البايتات ويرسلها إلى المخدم. يقوم المخدم باستلام رزمة البيانات وعرض المعلومات المرسله التي تحويها هذه الرزمة و من ثم إعادتها إلى الزبون.

يقوم الزبون عند استلام رزمة البيانات بعرض المعلومات التي تتضمنها. تكون في هذا المثال طريقة تنفيذ الترميز للصفوف PacketClientForm و PacketServerForm متشابهة.

المخدم

نعرض فيما يلي الكود الخاص بالصف PacketServerForm:

```
1 // PacketServer.cs
2 // Set up a server that will receive packets from a
3 // client and send the packets back to the client.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 using System.Threading;
9
10 public partial class PacketServerForm : Form
11 {
12     public PacketServerForm()
13     {
14         InitializeComponent();
15     } // end constructor
16
17     private UdpClient client;
18     private IPEndPoint receivePoint;
19
20     // initialize variables and thread for receiving packets
21     private void PacketServerForm_Load( object sender, EventArgs e )
22     {
23         client = new UdpClient( 50000 );
24         receivePoint = new IPEndPoint( new IPAddress( 0 ), 0 );
25         Thread readThread =
26             new Thread( new ThreadStart( WaitForPackets ) );
27         readThread.Start();
28     } // end method PacketServerForm_Load
29
30     // shut down the server
31     private void PacketServerForm_FormClosing( object sender,
32         FormClosingEventArgs e )
33     {
34         System.Environment.Exit( System.Environment.ExitCode );
35     } // end method PacketServerForm_FormClosing
36
```

```

37 // delegate that allows method DisplayMessage to be called
38 // in the thread that creates and maintains the GUI
39 private delegate void DisplayDelegate( string message );
40
41 // method DisplayMessage sets displayTextBox's Text property
42 // in a thread-safe manner
43 private void DisplayMessage( string message )
44 {
45     // if modifying displayTextBox is not thread safe
46     if ( displayTextBox.InvokeRequired )
47     {
48         // use inherited method Invoke to execute DisplayMessage
49         // via a delegate
50         Invoke( new DisplayDelegate( DisplayMessage ),
51             new object[] { message } );
52     } // end if
53     else // OK to modify displayTextBox in current thread
54         displayTextBox.Text += message;
55 } // end method DisplayMessage
56
57 // wait for a packet to arrive
58 public void WaitForPackets()
59 {
60     while ( true )
61     {
62         // set up packet
63         byte[] data = client.Receive( ref receivePoint );
64         DisplayMessage( "\r\nPacket received:" +
65             "\r\nLength: " + data.Length +
66             "\r\nContaining: " +
67             System.Text.Encoding.ASCII.GetString( data ) );
68
69         // echo information from packet back to client
70         DisplayMessage( "\r\n\r\nEcho data back to client..." );
71         client.Send( data, data.Length, receivePoint );
72         DisplayMessage( "\r\nPacket sent\r\n" );
73     } // end while
74 } // end method WaitForPackets
75 } // end class PacketServerForm
76

```

يقوم معالج الحدث Load في السطر 23 من الصف PacketServerForm ببناء متغير من نوع `UdpClient` المسؤول عن استلام البيانات على المنفذ 50000. تقوم هذه العملية بتهيئة المقبس (Socket) لتنفيذ عملية الاتصال.

يقوم السطر 24 ببناء متغير من الصف `IPEndPoint` ليحتفظ بعنوان IP و رقم المنفذ الذي يتم من خلاله إرسال البيانات إلى البرنامج PacketServerForm من قبل الزبائن.

يكون المعامل الأول لباني `IPEndPoint` غرض من الصف `IPAddress` أما المعامل الثاني فهو رقم المنفذ. يكون للمعاملات السابقة القيمة 0 لأننا نقوم بتهيئة غرض `IPEndPoint` فارغ. عند

وصول برقية معطيات من الزبون إلى المخدم فإنه يتم نسخ العنوان IP ورقم المنفذ إلى الغرض **IPEndPoint**.

يتم تعريف الطرائق **DisplayDelegate** و **DisplayMessage** في الأسطر 39-55 لإعطاء أي نيسب القدرة على تعديل الخاصية **Text** للعنصر **DisplayTextBox**.

تقوم الطريقة **WaitForPackets** بتنفيذ حلقة لا نهائية يتم من خلالها انتظار وصول البيانات إلى التطبيق. عند وصول البيانات تقوم الطريقة **Receive** (السطر 64) من الـ **UdpClient** باستلام سلسلة من البايتات من الزبون.

يتم تمرير الغرض **IPEndPoint** للطريقة **Receive** والذي تم إنشائه عن طريق الباني. وبهـ ذا يمكن للطريقة **Receive** نسخ عنوان IP ورقم المنفذ إلى الغرض **IPEndPoint** الممرر. وبهـ ذا الأسلوب يمكن ترجمة وتنفيذ البرنامج بدون أن يعطي أي استثناء (**Exception**) لأن الطريقة **Receive** تقوم بتهيئة الغرض **IPEndPoint** وإعطائه قيمة لا تساوي **Null**.

تقوم الأسطر 64-67 بتعديل واجهة الإظهار لتتضمن بيانات رزم البيانات المرسل.

يقوم السطر 71 بإعادة البيانات المرسل إلى الـ **UdpClient** ون بالـ استخدام الطريقة **Send** للـ **UdpClient**. تملك الطريقة **Send** ثلاثة معاملات:

- سلسلة البايتات المعدة للإرسال، رقم من نمط **int** يمثل طول السلسلة.
- والغرض **IPEndPoint** الذي يمثل وجهة الإرسال.

نستخدم المصفوفة **data** التي تُعيدها الطريقة **Receive** كمعامل أول للطريقة **Send** ونُمثل البيانات المعدة للإرسال، أما طول المصفوفة **data** فهو المعامل الثاني، أما المعامل الثالث فهـ و **IPEndPoint** الممررة للطريقة **Receive**.

يتم تخزين عنوان IP ورقم المنفذ للزبون المرسل في الغرض **ReceivePoint**، وذلك عن طريق تمرير هذا الغرض إلى الطريقة **Send** مما يسمح للبرنامج **PacketServerForm** بالاسـتجابة والرد على الزبون.

نعرض فيما يلي الكود الخاص بالصف `PacketClientForm`:

```

1 // PacketClient.cs
2 // Set up a client that sends packets to a server and receives
3 // packets from a server.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 Using System.Threading;
9
10 public partial class PacketClientForm : Form
11 {
12     public PacketClientForm()
13     {
14         InitializeComponent();
15     } // end constructor
16
17     private UdpClient client;
18     private IPEndPoint receivePoint;
19
20     // initialize variables and thread for receiving packets
21     private void PacketClientForm_Load( object sender, EventArgs e )
22     {
23         receivePoint = new IPEndPoint( new IPAddress( 0 ), 0 );
24         client = new UdpClient( 50001 );
25         Thread thread =
26             new Thread( new ThreadStart( WaitForPackets ) );
27         thread.Start();
28     } // end method PacketClientForm_Load
29
30     // shut down the client
31     private void PacketClientForm_FormClosing( object sender,
32         FormClosingEventArgs e )
33     {
34         System.Environment.Exit( System.Environment.ExitCode );
35     } // end method PacketClientForm_FormClosing
36
37     // delegate that allows method DisplayMessage to be called
38     // in the thread that creates and maintains the GUI
39     private delegate void DisplayDelegate( string message );
40
41     // method DisplayMessage sets displayTextBox's Text property
42     // in a thread-safe manner
43     private void DisplayMessage( string message )
44     {
45         // if modifying displayTextBox is not thread safe
46         if ( displayTextBox.InvokeRequired )
47         {
48             // use inherited method Invoke to execute DisplayMessage
49             // via a delegate
50             Invoke( new DisplayDelegate( DisplayMessage ),
51                 new object[] { message } );

```

```

52     } // end if
53     else // OK to modify displayTextBox in current thread
54         displayTextBox.Text += message;
55 } // end method DisplayMessage
56
57 // send a packet
58 private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
59 {
60     if ( e.KeyCode == Keys.Enter )
61     {
62         // create packet (datagram) as string
63         string packet = inputTextBox.Text;
64         displayTextBox.Text +=
65             "\r\nSending packet containing: " + packet;
66
67         // convert packet to byte array
68         byte[] data = System.Text.Encoding.ASCII.GetBytes( packet );
69
70         // send packet to server on port 50000
71         client.Send( data, data.Length, "127.0.0.1", 50000 );
72         displayTextBox.Text += "\r\nPacket sent\r\n";
73         inputTextBox.Clear();
74     } // end if
75 } // end method inputTextBox_KeyDown
76
77 // wait for packets to arrive
78 public void WaitForPackets()
79 {
80     while ( true )
81     {
82         // receive byte array from server
83         byte[] data = client.Receive( ref receivePoint );
84
85         // output packet data to TextBox
86         DisplayMessage( "\r\nPacket received:" +
87             "\r\nLength: " + data.Length + "\r\nContaining: " +
88             System.Text.Encoding.ASCII.GetString( data ) + "\r\n" );
89     } // end while
90 } // end method WaitForPackets
91 } // end class PacketClientForm

```

يعمل الصف **PacketClientForm** بطريقة مشابهة للصف **PacketServerForm** - كما أن الغرض **Client** يقوم بإرسال رزم البيانات فقط وذلك عندما يقوم المستخدم بكتابة رسالة في مربع النص ومن ثم ضغط المفتاح **Enter**. إذ يضغط المستخدم على زر **inputTextBox_KeyDown** (الأسطر 58-75).

يقوم السطر 68 بتحويل السلسلة المحرفية التي أدخلها المستخدم في مربع نص إلى مصفوفة من البايتات. يستدعي السطر 71 الطريقة **Send** من الصف **UdpClient** لإرسال مصفوفة البايتات إلى البرنامج **PacketServerForm**.

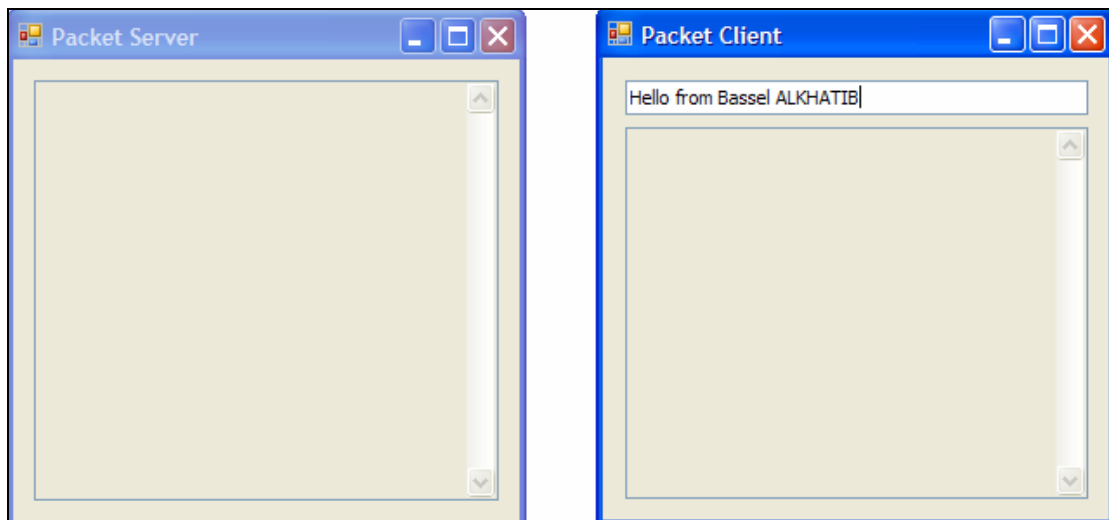
يتم تعريف الطرائق `DisplayMessage` و `DisplayDelegate` في الأسطر 39-55 لإعطاء أي
نيسب القدرة على تعديل الخاصية `Text` للعنصر `DisplayTextBox`.

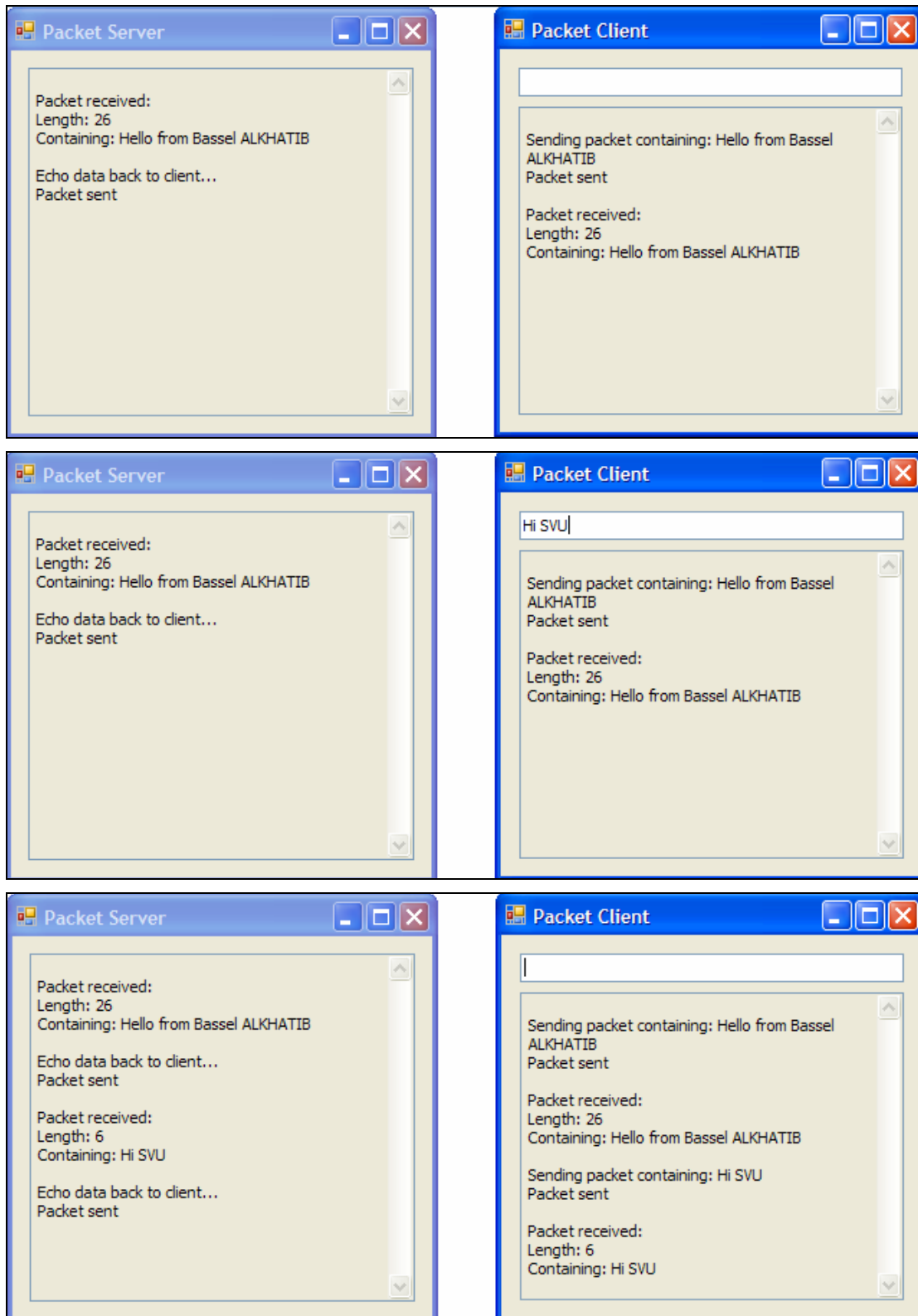
يقوم السطر 24 بتهيئة الغرض `UdpClient` لاستقبال رزم البيانات على المنفذ 50001 (استخدمنا
رقم المنفذ 50001 لأن `PacketServerForm` حجز رقم المنفذ 50000 مسبقاً).

تستخدم الطريقة `WaitForPackets` من الصف `PacketClientForm` (الأسطر 78-90) حلقة لا
نهائية لانتظار رزم البيانات المرسلة. يتم تجميد الطريقة `Receive` من الصف `UdpClient` حتى
يتم استقبال رزمة بيانات (السطر 83). إلا أن التجميد المنجز من قبل الطريقة `Receive` لا يمنع
الصف `PacketClientForm` من تنفيذ خدمات أخرى لأن نيسب آخر - رية - وم بتنفيذ الطريقة
`.WaitForPackets`.

عند وصول رزمة البيانات، تعرض الأسطر 68-88 مكونات هذه الرزمة ضمن مربع النص.
يمكن للمستخدم كتابة المعلومات المراد إرسالها إلى تطبيق المخدم في مربع النص ثم يـ م - وم
بضغط المفتاح `Enter` في أي وقت حتى في حال وصول رسالة من المخدم.

تُبين الأشكال التالية مختلف مراحل تفاعل المخدم والزيبون:





8. بناء تطبيق زبون/مخدم باستخدام مخدم متعدد النيات

نقوم في هذه الفقرة بدراسة وعرض كيفية تصميم نسخة شبكية من اللعبة الشائعة Tic-Tac-Toe (والتي تُعرف أيضاً بلعبة XO) وذلك عن طريق استخدام تقانة مقابس تدفق البيانات في بيئة زبون/مخدم.

يتكون البرنامج بشكل أساسي من الـ TicTacToeServer للمخدم، ومن الـ TicTacToeClient للزبون. يسمح صف المخدم TicTacToeServer لمتنسخين من صف الـ TicTacToeClient بالاتصال بالمخدم وباللعب مع بعضهما البعض.

يقوم المخدم كما سنرى ببناء متنسخ من الصف Player يسمح بمعالجة اتصال كل زبون وتنفيذه ضمن نيسب منفصل. مما يمكّن المخدم من معالجة الطلبات المرسلة من الزبونين.

يقوم المخدم بتحديد القيمة "X" للزبون الأول المتصل (اللاعب X يقوم بالحركة الأولى)، ثم تحديد القيمة "O" للزبون الثاني.

يحتفظ المخدم أثناء عملية اللعب بمعلومات تحوي حالة واجهة تطبيق الزبونين ليتمكن المخدم من اختبار صحة حركة اللاعبين. يمكن لكل من المخدم والزبون معرفة اللاعب الفائز. (تركنا في هذا المثال الطريقة GameOver تُعيد دوماً القيمة False، أي لم نضع الكود اللازم لتحديد فوز أحد الزبونين).

يقوم كل زبون بعرض واجهة اللعبة الخاصة به ويقوم فقط بوضع علامة في مربع فارغ على اللوحة في واجهة المستخدم. يستخدم الصف Square لتعريف المربعات على لوحة اللعب.

المخدم

نعرض فيما يلي الكود الخاص بالصف TicTacToeServerForm:

```
1 // TicTacToeServer.cs
2 // This class maintains a game of Tic-Tac-Toe for two
3 // client applications.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 using System.Threading;
9 using System.IO;
10
11 public partial class TicTacToeServerForm : Form
12 {
13     public TicTacToeServerForm()
14     {
15         InitializeComponent();
```

```

16     } // end constructor
17
18 private byte[] board; //the local representation of the game board
19 private Player[] players; // two Player objects
20 private Thread[] playerThreads; // Threads for client interaction
21 private TcpListener listener; // listen for client connection
22 private int currentPlayer; // keep track of whose turn it is
23 private Thread getPlayers; // Thread acquiring client connections
24 internal bool disconnected = false; // true if the server closes
25
26 // initialize variables and thread for receiving clients
27 private void TicTacToeServerForm_Load(object sender, EventArgs e )
28     {
29         board = new byte[ 9 ];
30         players = new Player[ 2 ];
31         playerThreads = new Thread[ 2 ];
32         currentPlayer = 0;
33
34         // accept connections on a different thread
35         getPlayers = new Thread( new ThreadStart( SetUp ) );
36         getPlayers.Start();
37     } // end method TicTacToeServerForm_Load
38
39 // notify Players to stop Running
40 private void TicTacToeServerForm_FormClosing( object sender,
41     FormClosingEventArgs e )
42     {
43         disconnected = true;
44         System.Environment.Exit( System.Environment.ExitCode );
45     } // end method TicTacToeServerForm_FormClosing
46
47 // delegate that allows method DisplayMessage to be called
48 // in the thread that creates and maintains the GUI
49 private delegate void DisplayDelegate( string message );
50
51 // method DisplayMessage sets displayTextBox's Text property
52 // in a thread-safe manner
53 internal void DisplayMessage( string message )
54     {
55         // if modifying displayTextBox is not thread safe
56         if ( displayTextBox.InvokeRequired )
57         {
58             // use inherited method Invoke to execute
59             // DisplayMessage via a delegate
60             Invoke( new DisplayDelegate( DisplayMessage ),
61                 new object[] { message } );
62         } // end if
63         else // OK to modify displayTextBox in current thread
64             displayTextBox.Text += message;
65     } // end method DisplayMessage
66
67 // accepts connections from 2 players
68 public void SetUp()
69     {
70         DisplayMessage( "Waiting for players...\r\n" );
71
72         // set up Socket
73         listener =
74         new TcpListener( IPAddress.Parse( "127.0.0.1" ), 50000 );

```

```

75     listener.Start();
76
77     // accept first player and start a player thread
78     players[0]=new Player(listener.AcceptSocket(), this, 0 );
79     playerThreads[ 0 ] =
80         new Thread( new ThreadStart( players[ 0 ].Run ) );
81     playerThreads[ 0 ].Start();
82
83     // accept second player and start another player thread
84     players[1]=new Player(listener.AcceptSocket(), this, 1 );
85     playerThreads[ 1 ] =
86         new Thread( new ThreadStart( players[ 1 ].Run ) );
87     playerThreads[ 1 ].Start();
88
89     //let the 1 player know that the 2 player has connected
90     lock ( players[ 0 ] )
91     {
92         players[ 0 ].threadSuspended = false;
93         Monitor.Pulse( players[ 0 ] );
94     } // end lock
95 } // end method SetUp
96
97 // determine if a move is valid
98 public bool ValidMove( int location, int player )
99 {
100     // prevent another thread from making a move
101     lock ( this )
102     {
103         // while it is not the current player's turn, wait
104         while ( player != currentPlayer )
105             Monitor.Wait( this );
106
107         // if the desired square is not occupied
108         if ( !IsOccupied( location ) )
109         {
110             // set the board to contain the current player's mark
111             board[ location ] = ( byte ) ( currentPlayer == 0 ?
112                 'X' : 'O' );
113
114             // set the currentPlayer to be the other player
115             currentPlayer = ( currentPlayer + 1 ) % 2;
116
117             // notify the other player of the move
118             players[ currentPlayer ].OtherPlayerMoved(location );
119
120             // alert the other player that it's time to move
121             Monitor.Pulse( this );
122             return true;
123         } // end if
124         else
125             return false;
126     } // end lock
127 } // end method ValidMove
128
129 // determines whether the specified square is occupied
130 public bool IsOccupied( int location )
131 {
132     if (board[ location ]== 'X' || board[ location ] == 'O' )
133         return true;

```

```

134         else
135             return false;
136     } // end method IsOccupied
137
138     // determines if the game is over
139     public bool GameOver()
140     {
141         // place code here to test for a winner of the game
142         return false;
143     } // end method GameOver
144 } // end class TicTacToeServerForm
145
146 // class Player represents a tic-tac-toe player
147 public class Player
148 {
149     internal Socket connection; // Socket for accepting a connection
150     private NetworkStream socketStream; // network data stream
151     private TicTacToeServerForm server; // reference to server
152     private BinaryWriter writer; // facilitates writing to stream
153     private BinaryReader reader; // facilitates reading from stream
154     private int number; // player number
155     private char mark; // player's mark on the board
156     internal bool threadSuspended=true; // if wait for other player
157
158     // constructor requiring Socket, TicTacToeServerForm and int
159     // objects as arguments
160     public Player( Socket socket, TicTacToeServerForm serverValue,
161         int newNumber )
162     {
163         mark = (newNumber == 0 ? 'X' : 'O');
164         connection = socket;
165         server = serverValue;
166         number = newNumber;
167
168         // create NetworkStream object for Socket
169         socketStream = new NetworkStream( connection );
170
171         // create Streams for reading/writing bytes
172         writer = new BinaryWriter( socketStream );
173         reader = new BinaryReader( socketStream );
174     } // end constructor
175
176     // signal other player of move
177     public void OtherPlayerMoved( int location )
178     {
179         // signal that opponent moved
180         writer.Write( "Opponent moved." );
181         writer.Write( location ); // send location of move
182     } // end method OtherPlayerMoved
183
184     // allows the players to make moves and receive moves
185     // from the other player
186     public void Run()
187     {
188         bool done = false;
189
190         // display on the server that a connection was made
191         server.DisplayMessage( "Player " + ( number == 0 ? 'X' : 'O' )
192             + " connected\r\n" );

```

```

193
194     // send the current player's mark to the client
195     writer.Write( mark );
196
197     // if number equals 0 then this player is X,
198     // otherwise 0 must wait for X's first move
199     writer.Write( "Player " + ( number == 0 ?
200     "X connected.\r\n" : "O connected, please wait.\r\n" ) );
201
202     // X must wait for another player to arrive
203     if ( mark == 'X' )
204     {
205         writer.Write( "Waiting for another player." );
206
207         // wait for notification from server that another
208         // player has connected
209         lock ( this )
210         {
211             while ( threadSuspended )
212                 Monitor.Wait( this );
213         } // end lock
214
215         writer.Write( "Other player connected. Your move." );
216     } // end if
217
218     // play game
219     while ( !done )
220     {
221         // wait for data to become available
222         while ( connection.Available == 0 )
223         {
224             Thread.Sleep( 1000 );
225
226             if ( server.disconnected )
227                 return;
228         } // end while
229
230         // receive data
231         int location = reader.ReadInt32();
232
233         // if the move is valid, display the move on the
234         // server and signal that the move is valid
235         if ( server.ValidMove( location, number ) )
236         {
237             server.DisplayMessage( "loc: " + location + "\r\n" );
238             writer.Write( "Valid move." );
239         } // end if
240         else // signal that the move is invalid
241             writer.Write( "Invalid move, try again." );
242
243         // if game over, set done to true to exit while loop
244         if ( server.GameOver() )
245             done = true;
246     } // end while loop
247
248     // close the socket connection
249     writer.Close();
250     reader.Close();
251     socketStream.Close();

```

```

252         connection.Close();
253     } // end method Run
254 } // end class Player

```

يستخدم الصف `TicTacToeServerForm` معالج الحدث `Load` لبناء مصفوفة `board` من البايتات لتخزين الحركات التي يقوم بها كل لاعب (السطر 29).

كما يقوم ببناء مصفوفة من عنصرين من النمط `Player` (السطر 30) ومصفوفة من عنصرين من النمط `Thread` (السطر 31). يمثل كل عنصر من المصفوفتين السابقتين لاعب في البرنامج. يتم تهيئة المتحول `currentPlayer` بالقيمة 0 (السطر 32) ويعبر هذا المتحول عن اللاعب "X" وهو الذي يقوم بالحركة الأولى.

تقوم الأسطر 35-36 ببناء وتفعيل النيسب `getPlayers` والذي سيستخدمه الصف `TicTacToeServerForm` لقبول طلبات الاتصال، وبهذا فلا يتم تجميد النيسب الحالي بينما يتم انتظار اللاعبين.

تُعرف الأسطر 49-65 كل من `DisplayDelegate` و `DisplayMessage` لتسمح لأي نيسب بالتعديل على الخاصية `Text` للعنصر `displayTextBox` وذلك بشكل آمن. تُعرف الطريقة `DisplayMessage` على أنها `internal` وبهذا يمكن استدعاءها فقط داخل طريقة من الصف `Player`.

ينفذ النيسب `getPlayer` الطريقة `SetUp` (الأسطر 68-95)، والتي تقوم ببناء الغرض `TcpListener`، والذي يقوم بدوره بالاستماع لطلبات الاتصال من اللاعب الأول واللاعب الثاني على المنفذ 50000 (الأسطر 73-75).

تقوم الأسطر 78 و 84 بتهيئة المصفوفة `Player` (من عنصرين) والتي تمثل اللاعبين. كما تقوم الأسطر 79-81 و 85-87 ببناء نيسبين يتم من خلالهما تنفيذ الطريقة `Run` لكل عنصر من المصفوفة `Player`.

يتم تمرير ثلاثة معاملات لباني الصف **Player** (الأسطر 68-174):

- المعامل الأول هو مرجع إلى الغرض **Socket**.
- والمعامل الثاني هو مرجع للغرض **TicTacToeServerForm**.
- والمعامل الثالث هو رقم من النمط **int** يمثل رقم اللاعب يمكن من خلال هـ-ذا ال-رقم استنتاج الخطوة التي لعبها اللاعب إن كانت "X" (قيمة المعامل 0) أو "O" (قيمة المعامل 1).

يستدعي **TicTacToeServerForm** الطريقة **Run** (الأسطر 186-253) بعد تهيئة الغرض **Player**. تُعلم الأسطر 191-200 المخدم بتأسيس اتصال ناجح وإرسال محرف إلى الزبون ليقوم بوضعه على اللوحة عند قيامه بحركة.

إذا تم تنفيذ الطريقة **Run** للاعب "X"، تُنفذ الأسطر 205-215 ليتم انتظار إتمام اتصال اللاعب الثاني.

تُعرف الأسطر 211-212 حلقة **while** تقوم بوقف تنفيذ نيسب اللاعب "X" حتى يفي-وم المخدم بإعلام اللاعب "O" بأنه متصل. يتم معرفة أن اللاعب متصل من خلال إسناد القيمة **False** إلى المتحول **threadSuspended** من الصف **Player** (السطر 92). عندما تصبح قيمة المتحول **threadSuspended** تساوي **False** يتم إنهاء الحلقة **while** في الأسطر 211-212.

تُنفذ الطريقة **Run** حلقة **while** في الأسطر 219-224 ممكنة المستخدم من اللعب. تنتظر كل دورة من هذه الحلقة من المستخدم أن يرسل رقم من النمط **int** محدداً مكان اللعب على اللوحة ليتم وضع "X" أو "O". عند وضع اللاعب العلامة على لوحة التطبيق، يتم اختبار صلاحيته هذه الحركة (لا يحوي المربع مسبقاً أي علامة).

يمكن ملاحظة أن الحلقة **while** تستمر في التنفيذ ما دامت قيمة المتحول المنطقية **done** هي **False**. يتم تعديل قيمة هـ-ذا المتحول إلى القيمة **True** عن طريق معالج الحدث **TicTacToeServerForm_FormClosing** والذي يتم استدعائه عند إغلاق المخدم للاتصال. يبدأ السطر 222 بتنفيذ حلقة **while** ويستمر تنفيذ هذه الحلقة حتى تشير الخاصية **Available** بأن هنالك معلومات يجب استلامها من المقيس. إذا لم يكن هنالك أية معلومات يذهب النيسب إلى الحالة **sleep** لمدة ثانية واحدة.

يستخدم النسيب عند إيقاظه الخاصية Disconnectd لفحص فيما إذا كانت قيمة متحول المخ-دم Disconnectd مساوية True. إذا كانت القيمة مساوية True يتم إنهاء النسيب وإلا تستمر الحلقة في التنفيذ.

إذا كانت قيمة الخاصية Available تشير إلى وجود معلومات يجب استقبالها، فعندها يتم إنهاء تنفيذ الحلقة while في الأسطر 222-228 ليتمكن البرنامج من معالجة هذه البيانات. تحوي هـ-ذه البيانات على رقم من النمط int يمثل المكان الذي اختاره الزبون لوضع العلامة.

يستدعي السطر 231 الطريقة Readint32 من الصف BinaryReader لتقوم بقراءة هذا ال-رقم. يقوم ال-سطر 235 بع-دها بتمري-ر هـ-ذا ال-رقم إلى الطريقة ValidMove م-ن ال-صف TicTacToeServerForm. تقوم هذه الطريقة بفحص صلاحية هذه الحركة، فإذا كانت ممكنة يقوم الصف Player بوضع العلامة في المكان المحدد. تُرسل الطريقة ValidMove (الأس-طر 127-98) رسالة إلى الزبون مشيرة إلى إمكانية هذه الحركة.

تُمثل الأماكن على اللوحة في واجهة المستخدم بالأرقام من 0 إلى 8. تُغاف كل تعليمات الطريقة ValidMove بالتعليمة lock والتي تسمح بمحاولة لعب حركة واحدة فقط في نفس الوقت.

هذا يمنع اللاعبين من تعديل معلومات الحالة الحالية للعبة بصورة متزامنة. عند محاولة لاء-ب فحص إمكانية الحركة وهو ليس اللاعب الحالي (أي ليس دوره)، يتم وضع هذا اللاعب في حالة انتظار ريثما يحين دوره للعب.

إذا حاول المستخدم وضع علامة في مكان مشغول سابقاً، تقوم الطريقة ValidMove بإع-ادة القيمة False. إذا قام المستخدم بوضع العلامة في مكان خال (السطر 108)، تقوم الأسطر 111-112 بوضع العلامة بتمثيلها الصحيح على لوح واجهة المستخدم. يعا-م ال-سطر 118 اللا-ب الآخر بإتمام الحركة من اللاعب الأول، كما يستدعي السطر 121 الطريقة Pluse بحيث يمكن اللاعب المنتظر من أن يقوم بالحركة.

الزبون

يقوم TicTacToeServerClient ببناء مربع نص لإظهار الرسائل المرسل م-ن قبل المخ-دم وإظهار اللعبة على اللوحة.

تتكون واجهة المستخدم من تسعة أغرض من الصف Square، وتتضمن هذه الأغراض أل-واح Panels والتي سيفوم المستخدم بالنقر عليها مشيراً إلى الموقع الذي سيقع عليه العلامة عند إجراء الحركة.

يقوم معالج الحدث Load للصف TicTacToeServerClient بفتح اتصال مع المخدم (السطر 50) ويحتفظ بمرجع للاتصال المرتبط مع الغرض NetworkStream من الصف TcpClient (السطر 51).

تبدأ الأسطر 56-57 بتشغيل نيسب لقراءة الرسائل المرسله من المخدم إلى الزبون.

يمرر المخدم الرسائل إلى الطريقة ProcessMessage (الأسطر 179-215). إذا كانت الرسالة تشير إلى إمكانية إجراء الحركة (السطر 184) يقوم الزبون بوضع العلامة في المربع الحالي (المربع الذي نقر عليه المستخدم) ثم يقوم بإعادة رسم الواجهة لإظهار التعديل. إذا كانت الرسالة تشير إلى عدم إمكانية القيام بالحركة (السطر 190) يطلب الزبون من المستخدم النقر على مربع آخر. إذا كانت الرسالة تشير إلى حركة مقابلة (السطر 197)، يقوم السطر 200 بقراءة رقم المخدم يحدد المكان على لوحة الزبون لوضع العلامة المقابلة.

```
1 // TicTacToeClient.cs
2 // Client for the TicTacToe program.
3 using System;
4 using System.Drawing;
5 using System.Windows.Forms;
6 using System.Net.Sockets;
7 using System.Threading;
8 using System.IO;
9
10 public partial class TicTacToeClientForm : Form
11 {
12     public TicTacToeClientForm()
13     {
14         InitializeComponent();
15     } // end constructor
16
17     private Square[ , ] board; // representation of the game board
18     private Square currentSquare; // the Square that this player chose
19     private Thread outputThread; // Thread for receiving data server
20     private TcpClient connection; // client to establish connection
21     private NetworkStream stream; // network data stream
22     private BinaryWriter writer; // facilitates writing to the stream
23     private BinaryReader reader; // facilitates reading the stream
24     private char myMark; // player's mark on the board
25     private bool myTurn; // is it this player's turn?
26     private SolidBrush brush; // brush for drawing X's and O's
27     private bool done = false; // true when game is over
28
29     // initialize variables and thread for connecting to server
30     private void TicTacToeClientForm_Load(object sender, EventArgs e)
31     {
32         board = new Square[ 3, 3 ];
33
34         // create 9 Square objects and place them on the board
35         board[ 0, 0 ] = new Square( board0Panel, ' ', 0 );
36         board[ 0, 1 ] = new Square( board1Panel, ' ', 1 );
37         board[ 0, 2 ] = new Square( board2Panel, ' ', 2 );
```

```

38     board[ 1, 0 ] = new Square( board3Panel, ' ', 3 );
39     board[ 1, 1 ] = new Square( board4Panel, ' ', 4 );
40     board[ 1, 2 ] = new Square( board5Panel, ' ', 5 );
41     board[ 2, 0 ] = new Square( board6Panel, ' ', 6 );
42     board[ 2, 1 ] = new Square( board7Panel, ' ', 7 );
43     board[ 2, 2 ] = new Square( board8Panel, ' ', 8 );
44
45     // create a SolidBrush for writing on the Squares
46     brush = new SolidBrush( Color.Black );
47
48     // make connection to server and get the associated
49     // network stream
50     connection = new TcpClient( "127.0.0.1", 50000 );
51     stream = connection.GetStream();
52     writer = new BinaryWriter( stream );
53     reader = new BinaryReader( stream );
54
55     // start a new thread for sending and receiving messages
56     outputThread = new Thread( new ThreadStart( Run ) );
57     outputThread.Start();
58 } // end method TicTacToeClientForm_Load
59
60 // repaint the Squares
61 private void TicTacToeClientForm_Paint( object sender,
62     PaintEventArgs e )
63 {
64     PaintSquares();
65 } // end method TicTacToeClientForm_Load
66
67 // game is over
68 private void TicTacToeClientForm_FormClosing( object sender,
69     FormClosingEventArgs e )
70 {
71     done = true;
72     System.Environment.Exit( System.Environment.ExitCode );
73 } // end TicTacToeClientForm_FormClosing
74
75 // delegate that allows method DisplayMessage to be called
76 // in the thread that creates and maintains the GUI
77 private delegate void DisplayDelegate( string message );
78
79 // method DisplayMessage sets displayTextBox's Text property
80 // in a thread-safe manner
81 private void DisplayMessage( string message )
82 {
83     // if modifying displayTextBox is not thread safe
84     if ( displayTextBox.InvokeRequired )
85     {
86         // use inherited method Invoke to execute DisplayMessage
87         // via a delegate
88         Invoke( new DisplayDelegate( DisplayMessage ),
89             new object[] { message } );
90     } // end if
91     else // OK to modify displayTextBox in current thread
92         displayTextBox.Text += message;
93 } // end method DisplayMessage
94
95 // delegate that allows method ChangeIdLabel to be called
96 // in the thread that creates and maintains the GUI

```

```

97 private delegate void ChangeIdLabelDelegate( string message );
98
99 // method ChangeIdLabel sets displayTextBox's Text property
100 // in a thread-safe manner
101 private void ChangeIdLabel( string label )
102 {
103     // if modifying idLabel is not thread safe
104     if ( idLabel.InvokeRequired )
105     {
106         // use inherited method Invoke to execute ChangeIdLabel
107         // via a delegate
108         Invoke( new ChangeIdLabelDelegate( ChangeIdLabel ),
109             new object[] { label } );
110     } // end if
111     else // OK to modify idLabel in current thread
112         idLabel.Text = label;
113 } // end method ChangeIdLabel
114
115 // draws the mark of each square
116 public void PaintSquares()
117 {
118     Graphics g;
119
120     // draw the appropriate mark on each panel
121     for ( int row = 0; row < 3; row++ )
122     {
123         for ( int column = 0; column < 3; column++ )
124         {
125             // get the Graphics for each Panel
126             g = board[ row, column ].SquarePanel.CreateGraphics();
127
128             // draw the appropriate letter on the panel
129             g.DrawString( board[ row, column ].Mark.ToString(),
130                 board0Panel.Font, brush, 10, 8 );
131         } // end for
132     } // end for
133 } // end method PaintSquares
134
135 // send location of the clicked square to server
136 private void square_MouseUp( object sender,
137     System.Windows.Forms.MouseEventArgs e )
138 {
139     // for each square check if that square was clicked
140     for ( int row = 0; row < 3; row++ )
141     {
142         for ( int column = 0; column < 3; column++ )
143         {
144             if ( board[ row, column ].SquarePanel == sender )
145             {
146                 CurrentSquare = board[ row, column ];
147
148                 // send the move to the server
149                 SendClickedSquare( board[ row, column ].Location );
150             } // end if
151         } // end for
152     } // end for
153 } // end method square_MouseUp
154
155 // control thread that allows continuous update of the

```

```

156 // TextBox display
157 public void Run()
158 {
159     // first get players's mark (X or O)
160     myMark = reader.ReadChar();
161     ChangeIdLabel( "You are player \"\" + myMark + \"\" );
162     myTurn = ( myMark == 'X' ? true : false );
163
164     // process incoming messages
165     try
166     {
167         // receive messages sent to client
168         while ( !done )
169             ProcessMessage( reader.ReadString() );
170     } // end try
171     catch ( IOException )
172     {
173         MessageBox.Show( "Server is down, game over", "Error",
174             MessageBoxButtons.OK, MessageBoxIcon.Error );
175     } // end catch
176 } // end method Run
177
178 // process messages sent to client
179 public void ProcessMessage( string message )
180 {
181     // if the move the player sent to the server is valid
182     // update the display, set that square's mark to be
183     // the mark of the current player and repaint the board
184     if ( message == "Valid move." )
185     {
186         DisplayMessage( "Valid move, please wait.\r\n" );
187         currentSquare.Mark = myMark;
188         PaintSquares();
189     } // end if
190     else if ( message == "Invalid move, try again." )
191     {
192         // if the move is invalid, display that and it is now
193         // this player's turn again
194         DisplayMessage( message + "\r\n" );
195         myTurn = true;
196     } // end else if
197     else if ( message == "Opponent moved." )
198     {
199         // if opponent moved, find location of their move
200         int location = reader.ReadInt32();
201
202         // set that square to have the opponents mark and
203         // repaint the board
204         board[ location / 3, location % 3 ].Mark =
205             ( myMark == 'X' ? 'O' : 'X' );
206         PaintSquares();
207
208         DisplayMessage( "Opponent moved. Your turn.\r\n" );
209
210         // it is now this player's turn
211         myTurn = true;
212     } // end else if
213     else
214         DisplayMessage( message + "\r\n" ); // display message

```

```

215 } // end method ProcessMessage
216
217 // sends the server the number of the clicked square
218 public void SendClickedSquare( int location )
219 {
220     // if it is the current player's move right now
221     if ( myTurn )
222     {
223         // send the location of the move to the server
224         writer.Write( location );
225
226         // it is now the other player's turn
227         myTurn = false;
228     } // end if
229 } // end method SendClickedSquare
230
231 // write-only property for the current square
232 public Square CurrentSquare
233 {
234     set
235     {
236         currentSquare = value;
237     } // end set
238 } // end property CurrentSquare
239 } // end class TicTacToeClientForm

```

```

1 // Square.cs
2 // A Square on the TicTacToe board.
3 using System.Windows.Forms;
4
5 // the representation of a square in a tic-tac-toe grid
6 public class Square
7 {
8     private Panel panel; // GUI Panel that represents this Square
9     private char mark; // player's mark on this Square (if any)
10    private int location; // location on the board of this Square
11
12    // constructor
13    public Square(Panel newPanel, char newMark, int newLocation )
14    {
15        panel = newPanel;
16        mark = newMark;
17        location = newLocation;
18    } // end constructor
19
20    // property SquarePanel; the panel which the square represents
21    public Panel SquarePanel
22    {
23        get
24        {
25            return panel;
26        } // end get
27    } // end property SquarePanel
28
29    // property Mark; the mark on the square
30    public char Mark
31    {
32        get

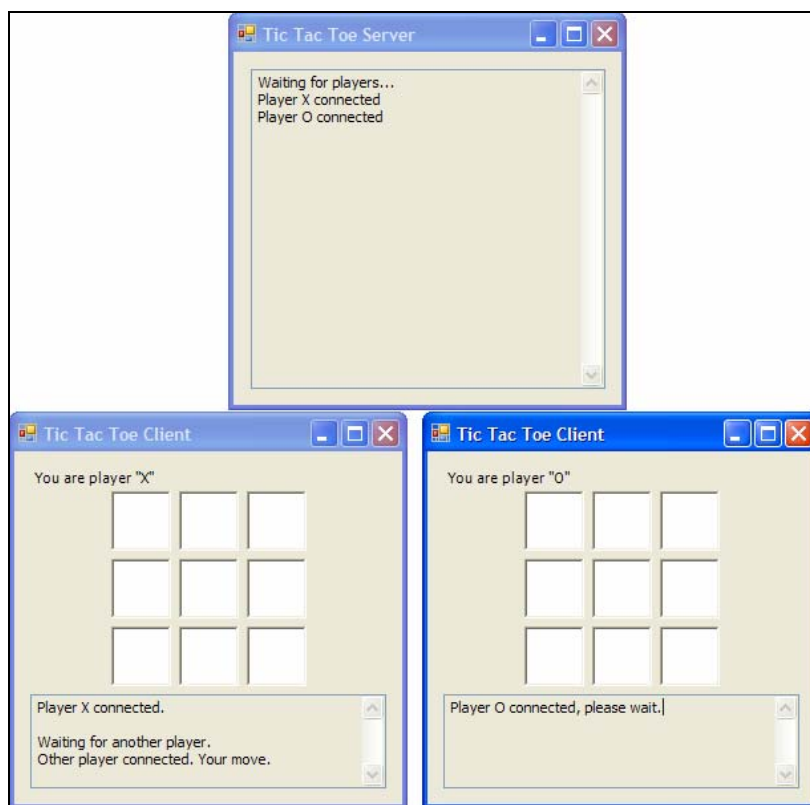
```

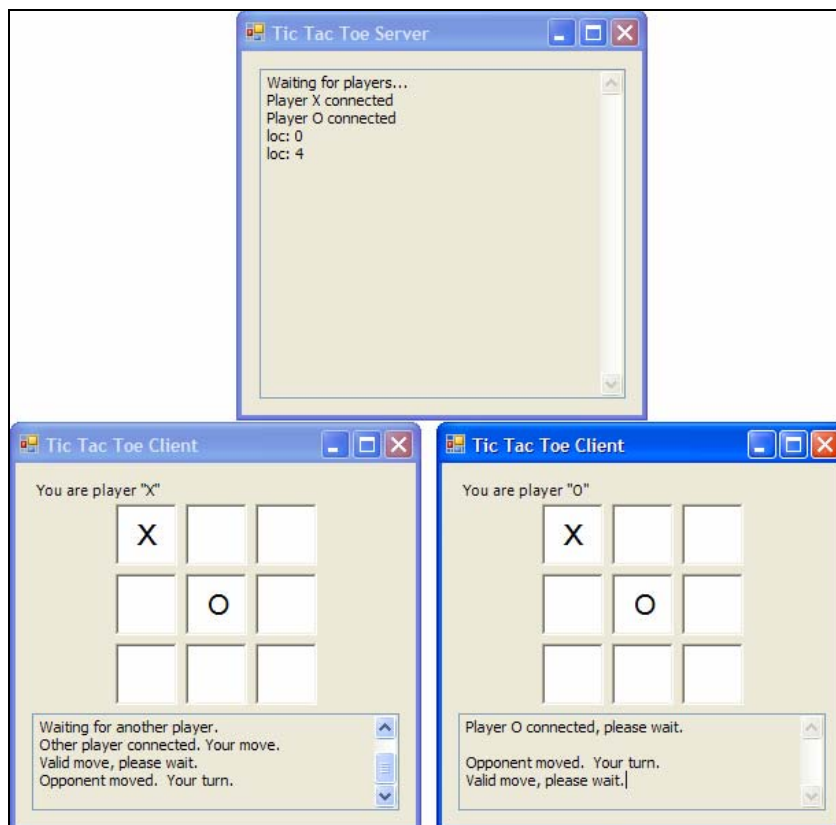
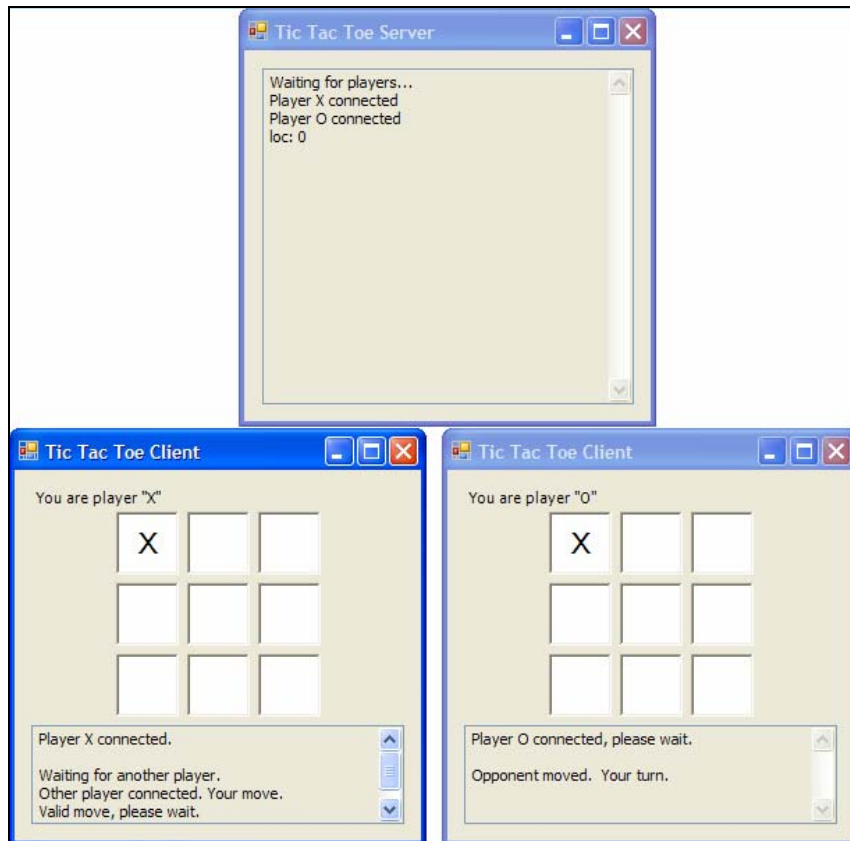
```

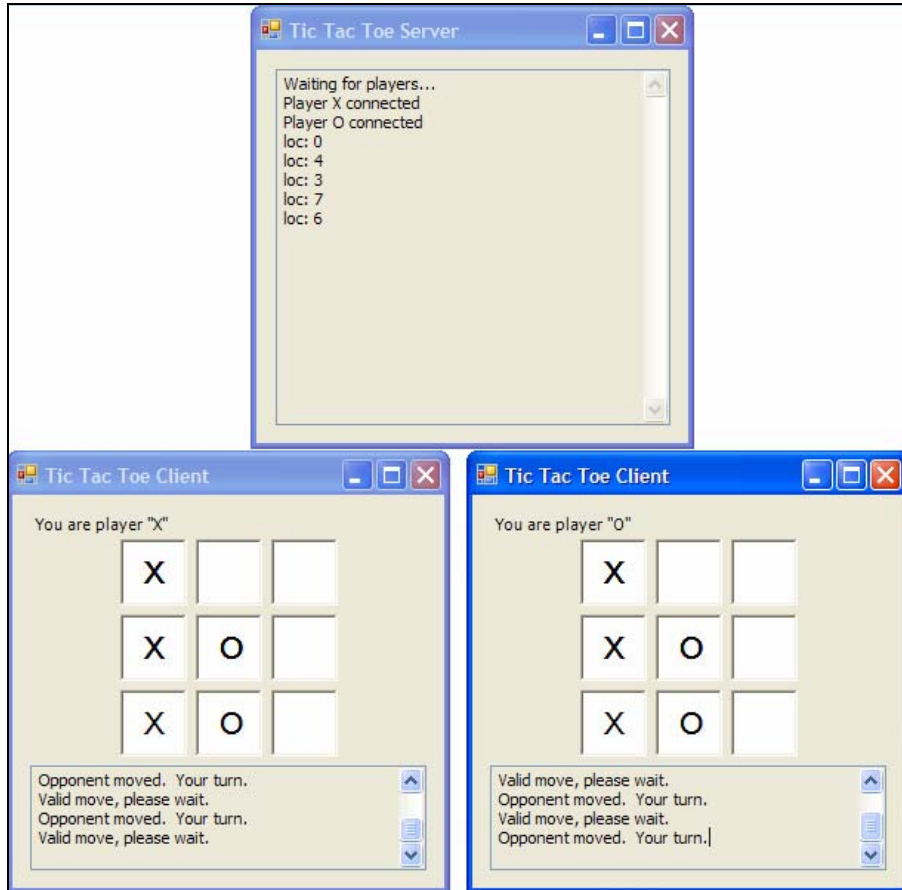
33     {
34         return mark;
35     } // end get
36     set
37     {
38         mark = value;
39     } // end set
40 } // end property Mark
41
42 // property Location; the square's location on the board
43 public int Location
44 {
45     get
46     {
47         return location;
48     } // end get
49 } // end property Location
50 } // end class Square

```

تُبين الأشكال التالية تنفيذ البرنامج السابق مع لاعبين:







9. عنصر التحكم WebBrowser

تتضمن FCL 2.0 عنصر التحكم WebBrowser والذي يمكّن التطبيق - ق م - ن دم - ج إمكانات استعراض الويب. يحوي عنصر التحكم WebBrowser طرائق لتصفح الوب والاحتفاظ بعناوين مواقع الويب التي يتصفحها المستخدم. كما يقدم إمكانية تفاعل المستخدم مع مكونات الصفحة المعروضة ضمن هذا العنصر. وبهذا يمكن للتطبيق الاستجابة والتفاعل مع المستخدم عند التفاعل على رابط ضمن صفحة الانترنت مثلاً.

يعرض الشكل التالي إمكانيات عنصر التحكم WebBrowser. يـ زود الـ صف BrowserForm الوظائف الأساسية لهذا العنصر.

مثلاً:

السماح للمستخدم باستعراض عنوان موقع على شبكة الإنترنت (URL) والتفـهـل بين الصفحات التي تمت زيارتها مسبقاً أو تحديث الصفحة الحالية.



```
1 // Browser.cs
2 // WebBrowser control example.
3 using System;
4 using System.Windows.Forms;
5
6 public partial class BrowserForm : Form
7 {
8     public BrowserForm()
9     {
10         InitializeComponent();
11     } // end constructor
12
13     // navigate back one page
14     private void backButton_Click( object sender, EventArgs e )
15     {
16         webBrowser.GoBack();
17     } // end method backButton_Click
18
19     // navigate forward one page
20     private void forwardButton_Click( object sender, EventArgs e )
```

```

21     {
22         webBrowser.GoForward();
23     } // end method forwardButton_Click
24
25     // stop loading the current page
26     private void stopButton_Click( object sender, EventArgs e )
27     {
28         webBrowser.Stop();
29     } // end method stopButton_Click
30
31     // reload the current page
32     private void reloadButton_Click( object sender, EventArgs e )
33     {
34         webBrowser.Refresh();
35     } // end method reloadButton_Click
36
37     // navigate to the user's home page
38     private void homeButton_Click( object sender, EventArgs e )
39     {
40         webBrowser.GoHome();
41     } // end method homeButton_Click
42
43     // if the user pressed enter, navigate to the specified URL
44     private void navigationTextBox_KeyDown( object sender,
45         KeyEventArgs e )
46     {
47         if ( e.KeyCode == Keys.Enter )
48             webBrowser.Navigate( navigationTextBox.Text );
49     } // end method navigationTextBox_KeyDown
50
51     // enable stopButton while the current page is loading
52     private void webBrowser_Navigating( object sender,
53         WebBrowserNavigatingEventArgs e )
54     {
55         stopButton.Enabled = true;
56     } // end method webBrowser_Navigating
57
58     // update the status text
59     private void webBrowser_StatusTextChanged( object sender,
60         EventArgs e )
61     {
62         statusTextBox.Text = webBrowser.StatusText;
63     } // end method webBrowser_StatusTextChanged
64
65     // update the ProgressBar
66     private void webBrowser_ProgressChanged( object sender,
67         WebBrowserProgressChangedEventArgs e )
68     {
69         pageProgressBar.Value =
70             (int) ((100 * e.CurrentProgress ) / e.MaximumProgress );
71     } // end method webBrowser_ProgressChanged
72
73     // update the web browser's controls appropriately
74     private void webBrowser_DocumentCompleted( object sender,
75         WebBrowserDocumentCompletedEventArgs e )
76     {
77         //set the text in navigationTextBox to the current page's URL
78         navigationTextBox.Text = webBrowser.Url.ToString();
79     }

```

```

80 // enable or disable backButton and forwardButton
81 backButton.Enabled = webBrowser.CanGoBack;
82 forwardButton.Enabled = webBrowser.CanGoForward;
83
84 // disable stopButton
85 stopButton.Enabled = false;
86
87 // clear the pageProgressBar
88 pageProgressBar.Value = 0;
89 } // end method webBrowser_DocumentCompleted
90
91 // update the title of the Browser
92 private void webBrowser_DocumentTitleChanged( object sender,
93 EventArgs e )
94 {
95     this.Text = webBrowser.DocumentTitle + " - Browser";
96 } // end method webBrowser_DocumentTitleChanged
97 } // end class BrowserForm

```

تُعرف الأسطر 14-41 معالج الحدث Click لكل من أزرار الاستعراض الخمسة الموجودة أعلى الشكل. يفوم كل معالج حدث باستدعاء طريقة من طرق عنصر التحكم WebBrowser الموافقة له.

- تنقل الطريقة GoBack (السطر 16) عنصر التحكم لاستعراض الصفحة السابقة.
 - تقوم الطريقة GoForward (السطر 22) بالانتقال إلى الصفحة التالية التي تمت زيارتها من قبل المستخدم.
 - توقف الطريقة Stop (السطر 28) تحميل الصفحة الحالية.
 - تُعيد الطريقة Refresh (السطر 34) تحميل الصفحة الحالية.
 - تقوم الطريقة GoHome باستعراض الصفحة الافتراضية.
- يسمح مربع النص الموجود إلى يمين أزرار التحكم للمستخدم بإدخال عنوان موقع على شبكة الإنترنت URL. عندما يقوم المستخدم بكتابة العنوان في هذا الحقل ومن ثم النقر على Enter، يتم تنفيذ معالج الحدث في الأسطر 44-49. يستدعي الأسطر 48 الطريقة Navigate لتحميل الصفحة وفق العنوان المدخل.
- يقوم WebBrowser بتوليد الحدث Navigating عندما يبدأ بتحميل صفحة جديدة، وعندها يتم تنفيذ معالج الحدث في الأسطر 52-56. يقوم السطر 55 بتنفيذ stopButton بحيث يمكن للمستخدم وقف تحميل الصفحة.
- عادةً، يستطيع المستخدم رؤية حالة تحميل الصفحة في أسفل واجهة المتصفح. لهذا السبب نضع العنصرين TextBox و ProgressBar في أسفل النموذج ولهما الأسماء StatusTextBox و

PageProgressBar على التوالي. يقوم **WebBrowser** بتوليد الحدث **StatusTextChanged** وذلك عند تغيير الخاصية **StatusText**، يقوم معالج الحدث الخاص بالحدث المولد (الأسطر 59-63) بوضع محتويات الخاصية **StatusText** ضمن الخاصية **Text** للعنصر **StatusTextBox** (السطر 62)، وبهذا يستطيع المستخدم مشاهدة الرسائل الناتجة عن حالة **WebBrowser**. يتم ومعالج الحدث **ProgressChanged** (الأسطر 66-71) بتحديث قيمة الخاصية **Value** للعنصر **PageProgressBar** ليعكس الحجم الذي تم تحميله من الصفحة المطلوبة.

عند انتهاء **WebBrowser** من تحميل الصفحة، يقوم بتوليد الحدث **DocumentCompleted**. ينتج عن ذلك تنفيذ معالج الحدث في الأسطر 74-89.

يقوم السطر 78 بتحديث محتويات العنصر **NavigationTextBox** بحيث يظهر العنوان الجاري تحميله وهذا ضروري خصوصاً في حال النقر من قبل المستخدم على ارتباط موجّه ودون من الصفحة الحالية.

تستخدم الأسطر 81-82 الخصائص **CanGoBack** و **CanGoForward** للتحكم بتفعيل وإلغاء تفعيل الأزرار أثناء استعراض الصفحات. عند إتمام تحميل الصفحة فإنه يتم إلغاء تفعيل الزر **StopButton**.

يقوم السطر 88 بإسناد القيمة 0 للخاصية **Value** لتشير إلى انتهاء تحميل الصفحة.

تُعرف الأسطر 92-96 معالج الحدث للحدث **DocumentTitleChanged** والذي يتم تنفيذه عند تحميل صفحة جديدة.

10. التشبيك عن بعد

تحتوي منصة .NET تقنيات معالجة موزعة تُدعى التشبيك عن بعد - **NET Remoting**. تسمح للبرنامج بالنفاذ إلى أغراض موجودة على حواسيب أخرى عبر الشبكة.

يشابه التشبيك عن بعد طريقة الاستدعاء عن بعد - **RMI Remote Method Invocation** في Java والاستدعاء الإجرائيات عن بعد - **RPC Remote Procedure Call** في لغات البرمجة الإجرائية. كما يشابه أيضاً خدمات الوب **Web Services** المعروضة في الفـصل القادم مع بعض الاختلافات. إذ يقوم تطبيق الزبون في خدمات الوب بالاتصال مع خدمة الوب المستضافة من قبل مخدم الوب.

يمكن أن يكتب الزبون وخدمة الويب باستخدام لغات برمجية مختلفة طالما يمكنهما إرسال رسائل باستخدام البروتوكول SOAP. أما في التشبيك عن بعد، فيقوم تطبيق الزبون بإجراء الاتصال مع تطبيق المخدم، إلا أن كل منهما يجب أن يكون مكتوباً باستخدام NET.

عند استخدام التشبيك عن بعد NET، يقوم كل من الزبون والمخدم بإجراء الاتصال بينهما عن طريق استدعاء الطرائق وإرسال الأغراض بين تطبيقيهما. تُدعى هذه العملية الإدارة بالأغراض Marshaling.

وتشمل:

1. القنوات Channels.
2. الإدارة بالأغراض Marshaling.
3. تطبيق.

القنوات Channels

يمكن للمخدم والزبون الاتصال وتبادل المعلومات باستخدام طريقة القنوات. تستخدم القنوات إما البروتوكول HTTP أو البروتوكول TCP لإرسال الرسائل. تمتاز قناة HTTP في أن جدران النار تسمح لها بالاتصال بشكل افتراضي، بينما تقوم بتجميد الاتصالات TCP غير المألوفة. إلا أن القناة TCP تُعتبر أفضل أداء من قناة HTTP.

يجب في تطبيق التشبيك عن بعد التي تقدمه NET أن يقوم كل من الزبون والمخدم بفتح قناة اتصال باستخدام البروتوكول نفسه للاتصال وتبادل المعطيات بينهما. نستخدم في مثالنا البروتوكول HTTP.

الإدارة بالأغراض Marshaling

يوجد طريقتين لإدارة غرض، وذلك عن طريق القيمة (by value) أو عن طريق المرجع (by reference).

تتطلب إدارة غرض عن طريق القيمة من الغرض أن يكون قابلاً للسلسلة Serializable. بحيث يوفر إمكانية تمثيله كرسالة موحدة يمكن أن تُرسل بين التطبيقات عبر الشبكة. ومن الجهة المستقبلية في الجهة المقابلة من القناة بإجراء العملية العكسية للسلسلة Deserialize للغرض لتشكيل نسخة من الغرض الأصلي.

ولتمكين غرض من أن يمتلك الخاصية Serialized و Deserialized يجب تزويد الـ الصف الخاص بهذا الغرض بالصفة [Serializable] أو تزويده بالواجهة Serializable (interface). تتطلب إدارة غرض عن طريق المرجع من الصف الخاص بالغرض التوسيع (extend) من الصف MarshalByRefObject من فضاء الأسماء System. يشار إلى الغرض المدار عن طريق المرجع بالغرض البعيد (remote object)، وإلى الـ صف الذي ينتمي إليه بالصف البعيد (remote class).

لا يتم إرسال الغرض نفسه عند استخدام الإدارة عن طريق المرجع، بل يتم بناء غرضين وكيلين يدعى الأول بالوكيل الشفاف transparent proxy والثاني بالوكيل الحقيقي real proxy. يزود الوكيل الأول الزبون بكل الخدمات العامة (public) للغرض البعيد.

فعلياً، عندما يستدعي الزبون طريقة أو خاصية من الوكيل الأول يبدو كما لو أنه يستدعيها من الغرض البعيد. وعندها يستدعي الوكيل الأول الطريقة Invoke للوكيل الثاني الذي يرسل الرسالة المناسبة من قناة الزبون إلى قناة المخدم. يستلم المخدم الرسالة ويقوم بتنفيذ الاستدعاء الطريقة المطلوبة أو الوصول إلى الخاصية المحددة للغرض الفعلي الموجود على المخدم.

تطبيق

يتألف التطبيق المثال التالي، والذي يهدف إلى عرض معلومات عن حالة الطقس باستخدام التشبيك عن بعد، من خمسة مكونات:

1. الصف القابل للسلسلة CityWeather والذي يمثل تقرير الطقس لمدينة واحدة.
2. الواجهة Report والتي تُعرف الخاصية Reports وهي غرض مدار يحتفظ بمجموعة من الأغراض من الصف CityWeather يتم تزويد تطبيق الزبون به.
3. الصف ReportInfo الذي يوسع الصف MarshalByRefObject ويرث الواجهة Report وسيتم تهيئته فقط في المخدم.
4. تطبيق المخدم WeatherServer الذي سيقوم بإعداد قناة المخدم وجعل الـ صف ReportInfo متاحاً ضمن عنوان URL خاص.
5. تطبيق الزبون WeatherClient الذي سيقوم بإعداد الزبون ويطلب غرض ReportInfo من المخدم للحصول على معلومات الطقس ليوم محدد.

يستخدم هذا التطبيق الموقع **Traveler's Foracast** التالي الذي يزود المعلومات عن حالة الطقس:

<http://iwin.nws.noaa.gov/iwin/us/traveler.html>

الصف CityWeather

يحتوي الصف **CityWeather** على معلومات الطقس لمدينة واحدة. يتم التصريح عن هذا الصف ضمن فضاء الأسماء **Weahter** (السطر 5)، كما وسيد شرخص من ملف مكتبة الصف **Weahter.dll** والتي يجب الإشارة إليها **reference** ضمن تطبيق المخدم وتطبيق الزبون.

ولهذا فيجب وضع هذا الصف (والواجهة **Report**) ضمن مشروع مكتبة الصف **class library** **project**. يتم التصريح عن الصف **CityWeather** باستخدام الوصفة **[Serializable]** (السطر 7) والتي تشير إلى أن الغرض من الصف **CityWeather** يمكن أن يدار عن طريق القيمة. وهو أمر ضروري لأن الأغراض من الصف **CityWeather** سيتم إعادتها عن طريق الخاصية **Reports** من الصف **ReportInfo**. يجب أن تتم إدارة القيم المعادة من الطرائق والخصائص المصرح عنها في الصف البعيد عن طريق القيمة عند إرسالها من المخدم إلى الزبون.

وبهذا فعندما يقوم الزبون باستدعاء طريقة النفاذ **get** التي تُعيد غرض **CityWeather**، ستقوم قناة المخدم بسلسلة **serialize** للغرض في رسالة، يمكن للزبون أن يعكس هذه السلسلة **deserialize** لبناء نسخة من الغرض الأصلي. يورث الصف **CityWeather** من الصف **IComparable** (السطر 8) بحيث يمكن للخاصية من النمط **ArrayList** التي ستحتوي أغراض من الصف **CityWeather** ترتيبها أبجدياً.

```
1 // CityWeather.cs
2 // Class representing the weather information for one city.
3 using System;
4
5 namespace Weather
6 {
7     [ Serializable ]
8     public class CityWeather : IComparable
9     {
10         private string cityName;
11         private string description;
12         private string temperature;
13
14         public CityWeather( string city, string information,
15             string degrees )
16         {
17             cityName = city;
18             description = information;
19             temperature = degrees;
20         } // end constructor
```

```

21
22 // read-only property that gets city's name
23 public string cityName
24 {
25     get
26     {
27         return cityName;
28     } // end get
29 } // end property cityName
30
31 // read-only property that gets city's weather description
32 public string Description
33 {
34     get
35     {
36         return description;
37     } // end get
38 } // end property Description
39
40 // read-only property that gets city's temperature
41 public string Temperature
42 {
43     get
44     {
45         return temperature;
46     } // end get
47 } // end property Temperature
48
49 // implementation of CompareTo method for alphabetizing
50 public int CompareTo( object other )
51 {
52     return string.Compare(
53         cityName, ( ( CityWeather ) other ).CityName );
54 } // end method Compare
55
56 // return string representation of this CityWeather object
57 // to display the weather report on the server console
58 public override string ToString()
59 {
60     return cityName + " | " + temperature + " | " + description;
61 } // end method ToString
62 } // end class CityWeather
63 } // end namespace Weather
64 }

```

يحتوي الصف `CityWeather` حقول (الأسطر 10-12) لتخزين اسم المدينة، درجة الحرارة الدنيا والعليا، ووصف حالة الطقس.

يقوم باني الصف `CityWeather` (الأسطر 14-20) بتهيئة هذه المتحولات.

تُعرف الأسطر 23-47 ثلاثة خصائص للقراءة فقط للسماح بالحصول على قيم الحقول الثلاثة السابقة. يستخدم الصف `CityWeather` الطريقة `CompareTo` من الواجهة `Comparable` وتأخذ هذه الطريقة مرجع لغرض وتعيد رقم من النمط `int` (الأسطر 50-54).

حيث نرغب بترتيب أغراض الصف CityWeather أبجدياً حسب اسم المدينة. لهـذا تـستدعي الطريقة CompareTo الطريقة Compare الخاصة بالنمط string وتقوم بتمرير CityName كمعامل دخل. يعيد الصف CityWeather تعريف الطريقة ToString لعرض معلومات المدينة (الأسطر 58-61).

تُستخدم الطريقة ToString من قبل تطبيق المخدم لعرض معلومات الطقس التي يحصل عليها من صفحة الوب Traveler's Foracast.

الواجهة Report

يتم التصريح عن هذه الواجهة ضمن فضاء الأسماء Weather (السطر 7) ويجب تضمينها مع الصف CityWeather في ملف مكتبة الصف Weather.Dll وبهذا يمكن استخدامها في تطبيق المخدم وتطبيق الزبون. تُصرح الواجهة Report عن خاصية للقراءة فقط مع الطريقة get الذي تُعيد قيمة من النمط ArrayList. تحوي هذه الخاصية أغراض من الـصف CityWeather. يستخدم التطبيق هذه الخاصية للحصول على المعلومات في تقرير الطقس: اسم المدينة، درجة الحرارة الدنيا والعليا، وحالة الطقس.

```
1 // Report.cs
2 // Interface that defines a property for getting
3 // the information in a weather report.
4 using System;
5 using System.Collections;
6
7 namespace Weather
8 {
9     public interface Report
10    {
11        ArrayList Reports
12        {
13            get;
14        } // end property Reports
15    } // end interface Report
16 } // end namespace Weather
```

الصف ReportInfo

يقوم الصف البعيد ReportInfo بوراثة الواجهة Report الـ 10 سطر في فصل الأسماء Weather. كما يوسع (extend) الصف MarshalByRefObject. يكون الصف ReportInfo جزءاً من تطبيق المخدم ولا يمكن الوصول إليه مباشرة من قبل تطبيق الزبون.

```
1 // ReportInfo.cs
2 // Class that implements interface Report, retrieves
3 // and returns data on weather
4 using System;
5 using System.Collections;
6 using System.IO;
7 using System.Net;
8 using Weather;
9
10 public class ReportInfo : MarshalByRefObject, Report
11 {
12     private ArrayList cityList; //cities,temperatures, descriptions
13
14     public ReportInfo()
15     {
16         cityList = new ArrayList();
17
18         // create WebClient to get access to Web page
19         WebClient myClient = new WebClient();
20
21         // get StreamReader for response so we can read page
22         StreamReader input = new StreamReader( myClient.OpenRead(
23             "http://iwin.nws.noaa.gov/iwin/us/traveler.html" ) );
24
25         string separator1="TAV12";//indicates first batch of cities
26         string separator2="TAV13";//indicates second batch of cities
27
28         // locate separator1 in Web page
29         while ( !input.ReadLine().StartsWith(separator1)); // do nothing
30         ReadCities( input ); // read the first batch of cities
31
32         // locate separator2 in Web page
33         while (!input.ReadLine().StartsWith(separator2)); // do nothing
34         ReadCities( input ); // read the second batch of cities
35
36         cityList.Sort(); //sort list of cities by alphabetical order
37         input.Close(); // close StreamReader to NWS server
38
39         // display the data on the server side
40         Console.WriteLine( "Data from NWS Web site:" );
41
42         foreach ( CityWeather city in cityList )
43         {
44             Console.WriteLine( city );
45         } // end foreach
46     } // end constructor
47
48     // utility method that reads a batch of cities
49     private void ReadCities( StreamReader input )
```

```

50 {
51     // day format and night format
52     string dayFormat =
53         "CITY          WEA      HI/LO    WEA      HI/LO";
54     string nightFormat =
55         "CITY          WEA      LO/HI    WEA      LO/HI";
56     string inputLine = "";
57
58     // locate header that begins weather information
59     do
60     {
61         inputLine = input.ReadLine();
62     } while ( !inputLine.Equals( dayFormat ) &&
63             !inputLine.Equals( nightFormat ) );
64
65     inputLine = input.ReadLine(); // get first city's data
66
67     // while there are more cities to read
68     while ( inputLine.Length > 28 )
69     {
70         // create CityWeather object for city
71         CityWeather weather = new CityWeather(
72             inputLine.Substring( 0, 16 ),
73             inputLine.Substring( 16, 7 ),
74             inputLine.Substring( 23, 7 ) );
75
76         cityList.Add( weather ); // add to ArrayList
77         inputLine = input.ReadLine(); // get next city's data
78     } // end while
79 } // end method ReadCities
80
81 // property for getting the cities' weather reports
82 public ArrayList Reports
83 {
84     get
85     {
86         return cityList;
87     } // end get
88 } // end property Reports
89 } // end class ReportInfo

```

تُعرف الأسطر 14-46 بانبي الصف **ReportInfo**. يقوم السطر 19 ببناء الغرض **WebClient** للتواصل مع مصدر المعلومات المعروف عن طريق عنوان **URL**. نستخدم في مثالنا عنوان **URL** لـ **NWS Traveler's Forecast** صفحة الانترنت - - - - - <http://iwin.nws.noaa.gov/iwin/us/traveler.html>.

تستدعي الأسطر 22-23 الطريقة **OpenRead** والتي تُعيد غرض من الغرض **Stream** الذي يستطيع البرنامج استخدامه في قراءة البيانات التي تحوي معلومات الطقس من العنوان السابق.

يستخدم `Stream` لبناء غرض من `الـ` `StreamReader` وبهـذا يمكن للبرنامج قراءة الصفحات المكتوبة بلغة `HTML` سطر سطر.

نهتم من صفحة الويب فقط بالقسم الذي يحوي جزئين من معلومات المدن `Albany` حتى `Reno` ومدينة `Lacke` حتى `Washington`. تبدأ معلومات الجزء الأول عند `د` `ظ` `هـ` `و` `ر` `الـ` `س` `لـ` `س` `لـ` `ة` `المحرفية` `"TAV12"`، أما الجزء الثاني فتبدأ معلوماته عند ظهور السلسلة المحرفية `"TAV13"`.

نقوم بتعريف المتحولين `separator1` و `separator2` لتخزين هاتين السلسلتين المحرفيتين. في السطر 29، يتم قراءة الصفحة المكتوبة بلغة `HTML` سطر في كل مرة ليتم العثور على `"TAV12"`. عند الوصول إلى `"TAV12"` يقوم البرنامج باستدعاء الطريقة `ReadCities` لتقوم بقراءة معلومات المدن ووضعها في الغرض `cityList`.

يقوم السطر 33 بقراءة الصفحة سطر في كل مرة ليتم العثور على `"TAV13"`. عند الوصول إلى `"TAV13"` يقوم السطر 34 باستدعاء آخر للطريقة `ReadCities` لتقوم بقراءة معلومات المدن. يقوم السطر 36 باستدعاء الطريقة `Sort` من الصف `ArrayList` لتقوم بترتيب الأغراض من الصف `CityWeather` أبجدياً حسب اسم المدينة. يقوم السطر 37 بإغلاق الاتصال مع الموقع المستخدم من قبل `StreamReader`. تقوم الأسطر 42-45 بعرض معلومات الطقس لكل مدينة على واجهة تطبيق المخدم.

تُصرح الأسطر 49-79 عن الطريقة `ReadCities` والتي تأخذ الغرض `StreamReader` كمعامل وتقوم بقراءة معلومات كل مدينة، ثم تقوم ببناء الغرض `CityWeather` لتخزين المعلومات ضمنها وتخزين الغرض الناتج ضمن `cityList`.

تستمر الحلقة `do ... while` (الأسطر 59-63) في قراءة الصفحة سطر كل مرة حتى تجد السطر الرئيسي الذي يعتبر ترويسة جدول الطقس.

يبدأ هذا السطر إما بتصميم نهاري (الأسطر 52-53) مشيراً إلى ترويسة معلومات فترة النهار. أو يبدأ بتصميم ليلي (الأسطر 54-55) مشيراً إلى ترويسة معلومات فترة الليل. يقوم السطر 65 بقراءة السطر التالي من الصفحة والذي يعتبر أول سطر يحوي معلومات الحرارة.

تقوم حلقة `while` (الأسطر 68-78) ببناء غرض جديد من الصف `CityWeather` الذي يمثل المدينة الحالية. تقوم هذه الحلقة بسبر السلسلة المحرفية التي تحوي معلومات الطقس للمدينة الحالية حيث تقوم بفصل اسم المدينة، حالة الطقس، ودرجة الحرارة. ثم يتم إضافة الغرض `CityWeather` إلى `cityList`.

بعدها، يتم قراءة سطر آخر ويتم تخزينه في `inputLine` من أجل المعالجة. تستمر هذه العملية مادام طول السطر الذي يتم قراءته من الصفحة أكبر من 28 (الأسطر التي تحتوي على معلومات الطقس طولها أكبر من 28 حرف).

تُعيد الخاصية `Reports` (الأسطر 82-88) `cityList`. يقوم تطبيق الزبون باستدعاء هذه الخاصية عن بعد للحصول على تقرير الطقس اليومي.

الصف WeatherServer

تُحدد الأسطر 5-7 فضاءات الأسماء المستخدمة في تطبيق المخدم حيث يحوي السطر 5 والسطر 6 فضاء الأسماء المستخدم في التشبيك عن بعد في .NET. أما السطر 7 فيستخدم من أجل فضاءات البروتوكول HTTP. يحدد السطر 8 فضاء الأسماء `Weather` والذي يحوي الواجهة `Report`.

```
1 // WeatherServer.cs
2 // Server application that uses .NET remoting to send
3 // weather report information to a client
4 using System;
5 using System.Runtime.Remoting;
6 using System.Runtime.Remoting.Channels;
7 using System.Runtime.Remoting.Channels.Http;
8 using Weather;

9 class WeatherServer
10 {
11     static void Main( string[] args )
12     {
13         // establish HTTP channel
14         HttpChannel channel = new HttpChannel( 50000 );
15         ChannelServices.RegisterChannel( channel, false );
16
17         // register ReportInfo class
18         RemotingConfiguration.RegisterWellKnownServiceType(
19             typeof( ReportInfo ), "Report",
20             WellKnownObjectMode.Singleton );
21
22         Console.WriteLine( "Press Enter to terminate server." );
23         Console.ReadLine();
24     } // end Main
25 } // end class WeatherServer
```

تقوم الأسطر 15-16 في الطريقة `Main` بتسجيل (register) قناة HTTP على الحاسوب الحالي على المنفذ 50000. سيستخدم الزبون رقم هذا المنفذ للاتصال مع `WeatherServer` عن بعد. يشير المعامل `False` عند السطر 16 إلى عدم الرغبة بتنفيذ الحماية. تقوم الأسطر 19-21 بتسجيل نمط الصف `ReportInfo` عند "Report" كصف بعيد من النمط `Singleton`. إذا سجل الصف البعيد على أنه `Singleton` فإنه سيتم بناء غرض بعيد واحد عندما يطلب الزبون الأول ذلك الصف البعيد. وهذا الغرض البعيد سيقوم بتخديم كل الزبائن.

إذا سجل الصف على أنه **SingleCall** سيتم بناء غرض لكل استدعاء طريقة عن بعد لذلك الصف البعيد.

لاتكون فترة حياة الغرض البعيد من النمط **Singleton** دائمة، بل يتم تحريره عن طريق جـامع النفايات (**Garbage Collector**) عند عدم تخدمه أي استدعاء لفترة 5 دقائق. وفي حال طلب الغرض من قبل الزبون في وقت لاحق يتم بناء غرض جديد من النمط **Singleton**.

يمكن طلب الصف **ReportInfo** الآن من قبل الزبون على العنوان **URL**: **http://IPAddress:50000/Report** حيث **IPAddress** هو عنوان IP للحاسوب الذي يشغل تطبيق الزبون. تبقى القناة مفتوحة مادام تطبيق المخدم قيد العمل. ينتظر الـسطر 24 من استخدام التطبيق المخدم ليقوم بالضغط على المفتاح **Enter** قبل إنهاء التطبيق.

الصف WeatherClientForm

وهو واجهة التطبيق التي تستخدم التشبيك عن بعد للحصول على معلومات الطقس من **WeatherServer** لعرض هذه المعلومات بصورة بيانية سهلة القراءة. تحتوي واجهة المستخدم على 43 عنصر من النوع **Label** لكل مدينة. تتوضع كل هذه العناصر على لوحة من النمط **Panel**.

```
1 // WeatherClient.cs
2 // Client that uses .NET remoting to retrieve a weather report.
3 using System;
4 using System.Collections;
5 using System.Drawing;
6 using System.Windows.Forms;
7 using System.Runtime.Remoting;
8 using System.Runtime.Remoting.Channels;
9 using System.Runtime.Remoting.Channels.Http;
10 using Weather;
11
12 public partial class WeatherClientForm : Form
13 {
14     public WeatherClientForm()
15     {
16         InitializeComponent();
17     } // end constructor
18
19     // retrieve weather data
20     private void WeatherClientForm_Load(object sender, EventArgs e)
21     {
22         //setup HTTP channel, does not need to provide a port number
23         HttpChannel channel = new HttpChannel();
24         ChannelServices.RegisterChannel( channel, false );
25
26         //obtain a proxy for an object that implements interface Report
```

```

27 Report info = ( Report ) RemotingServices.Connect(
28     typeof( Report ), "http://localhost:50000/Report" );
29
30 // retrieve an ArrayList of CityWeather objects
31 ArrayList cities = info.Reports;
32
33 // create array and populate it with every Label
34 Label[] cityLabels = new Label[ 43 ];
35 int labelCounter = 0;
36
37 foreach ( Control control in displayPanel.Controls )
38 {
39     if ( control is Label )
40     {
41         cityLabels[ labelCounter ] = ( Label ) control;
42         ++labelCounter; // increment Label counter
43     } // end if
44 } // end foreach
45
46 // create Hashtable and populate with all weather conditions
47 Hashtable weather = new Hashtable();
48 weather.Add( "SUNNY", "sunny" );
49 weather.Add( "PTCLDY", "pcloudy" );
50 weather.Add( "CLOUDY", "mcloudy" );
51 weather.Add( "MOCLDY", "mcloudy" );
52 weather.Add( "TSTRMS", "rain" );
53 weather.Add( "RAIN", "rain" );
54 weather.Add( "SNOW", "snow" );
55 weather.Add( "VRYHOT", "vryhot" );
56 weather.Add( "FAIR", "fair" );
57 weather.Add( "RNSNOW", "rnsnow" );
58 weather.Add( "SHWRS", "showers" );
59 weather.Add( "WINDY", "windy" );
60 weather.Add( "NOINFO", "noinfo" );
61 weather.Add( "MISG", "noinfo" );
62 weather.Add( "DRZL", "rain" );
63 weather.Add( "HAZE", "noinfo" );
64 weather.Add( "SMOKE", "mcloudy" );
65 weather.Add( "SNOWSHWRS", "snow" );
66 weather.Add( "FLRRYS", "snow" );
67 weather.Add( "FOG", "noinfo" );
68
69 // create the font for the text output
70 Font font = new Font( "Courier New", 8, FontStyle.Bold );
71
72 // for every city
73 for ( int i = 0; i < cities.Count; i++ )
74 {
75     // use array cityLabels to find the next Label
76     Label currentCity = cityLabels[ i ];
77
78     //use ArrayList cities to find the next CityWeather object
79     CityWeather city = ( CityWeather ) cities[ i ];
80
81     // set current Label's image to image
82     // corresponding to the city's weather condition -
83     // find correct image name in Hashtable weather
84     currentCity.Image = new Bitmap( @"images\" +
85         weather[ city.Description.Trim() ] + ".png" );

```

```

86         currentCity.Font = font; // set font of Label
87         currentCity.ForeColor = Color.White;
88
89         // set Label's text to city name and temperature
90         currentCity.Text="\r\n " + city.CityName + city.Temperature;
91     } // end for
92 } // end method WeatherClientForm_Load
93 } // end class WeatherClientForm

```

تقوم الطريقة WeatherClientForm_Load (الأسطر 20-92) عند تحميل واجهة التطبيق بالحصول على معلومات الطقس. يقوم السطر 23 ببناء قناة HTTP بدون تحديد رقم المنفذ. هذا يجعل باني الصف HTTPChannal يقوم باختيار أي رقم منفذ متوفر على حاسب الزبون.

لا يكون رقم المنفذ ضرورياً لأن هذا التطبيق لا يمتلك الزبون الخاص به الذي يحتاج أن يعرف رقم المنفذ. يسجل السطر 24 القناة على حاسب الزبون. هذا يسمح للمخدم بإعادة إرسال المعلومات إلى الزبون.

تقوم الأسطر 27-28 بتعريف المتحول Report ويقوم بالإسناد له الوكيل للغرض Report الذي تمت تهيئته من قبل المخدم. يسمح الوكيل للزبون بالاستدعاء عن بعد خصائص الصف ReportInfo عن طريق تحويل الاستدعاءات إلى المخدم. تقوم الطريق Connect من الصف RemotingServices بالاتصال بالمخدم وإعادة مرجع إلى وكيل الغرض Report.

نقوم هنا بتنفيذ تطبيق المخدم وتطبيق الزبون على نفس الحاسب ولهذا نستخدم localhost في عنوان URL الذي يمثل تطبيق المخدم. للاتصال بتطبيق WeatherServer يتم تنفيذ هذه على حاسب مختلف، يجب تعديل عنوان URL السابق. يقوم السطر 31 بالحصول على المصفوفة ArrayList من الأغراض CityWeather التي تولد من قبل باني الصف ReportInfo (الأسطر 14-64). يشير المتحول cities إلى مصفوفة الأغراض CityWeather التي تحوي المعلومات من الصفحة Traveler's Forcast.

بما أن التطبيق يقوم بعرض معلومات الطقس لعدة مدن، يجب الانتباه إلى طريقة تنظيم المعلومات على اللواصق (Labels) وضمان أن كل وصف لمعلومات الطقس موافق للصورة المناسبة.

يستخدم البرنامج مصفوفة لتخزين اللواصق وغرض من الصف Hashtable لتخزين وصف الطقس وأسماء الصور الموافقة للوصف.

يقوم السطر 34 ببناء مصفوفة مراجع للواصق.

تضع الأسطر 35-44 اللواصق التي تم بناؤها في واجهة التصميم في مـ. صفوفه وبهـ. ذا يمكن الوصول إليها برمجياً لعرض معلومات الطقس لكل مدينة.

يقوم السطر 47 ببناء غرض من الصف Hashtable والذي يقوم بتخزين الزوج حالة الطقس - الصورة الموافقة لهذه الحالة.

تقوم الأسطر 73-91 بتهيئة اللواصق بحيث تحوي اسم المدينة، درجة الحرارة الحالية والصورة الموافقة لحالة الطقس لهذه المدينة.

يقوم السطر 76 بالحصول على اللاصقة من أجل عرض معلومات الطقس للمدينة التالية.

يستخدم السطر 79 المصفوفة cities للحصول على الأغـ. راض CityWeather التي تحوي معلومات الطقس لكل مدينة.

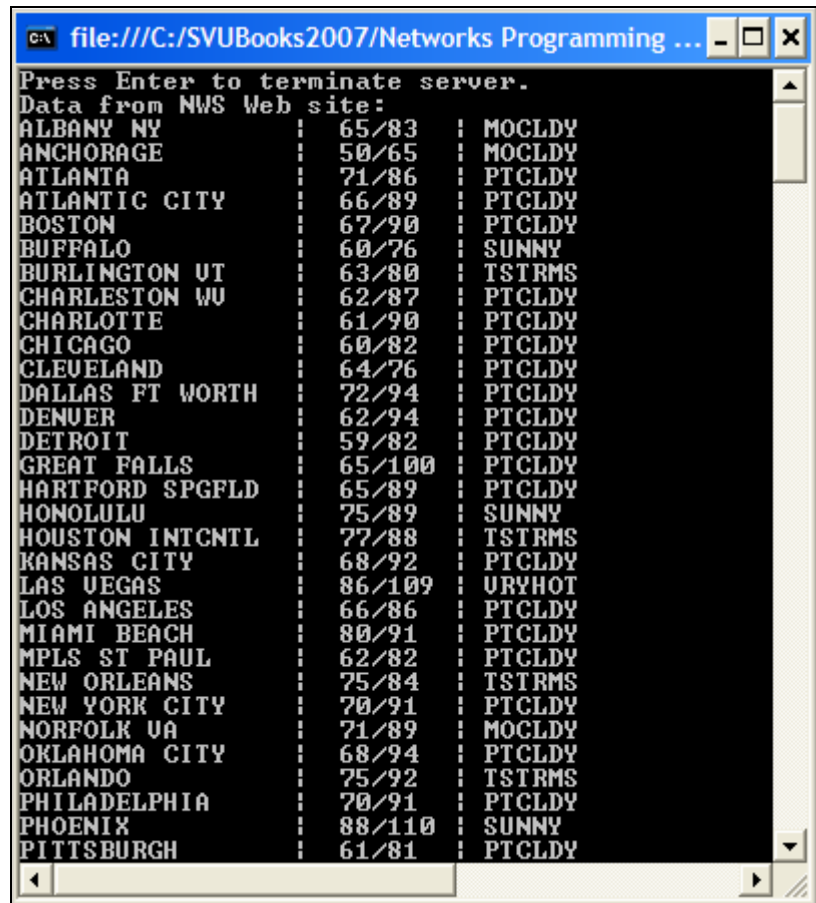
تقوم الأسطر 86-87 بتهيئة خصائص اللواصق ليتم عرض التأثيرات المرئية.

يقوم السطر 90 بتهيئة الخاصية Text لعرض اسم المدينة ودرجة الحرارة الدنيا والعليا.

يبين الشكل التالي واجهة الزبون:



يبين الشكل التالي واجهة المخدم:



The screenshot shows a Windows command prompt window with the title bar "C:\ file:///C:/SVUBooks2007/Networks Programming ...". The window contains the following text:

```
Press Enter to terminate server.  
Data from NWS Web site:  
ALBANY NY : 65/83 : MOCLDY  
ANCHORAGE : 50/65 : MOCLDY  
ATLANTA : 71/86 : PTCLDY  
ATLANTIC CITY : 66/89 : PTCLDY  
BOSTON : 67/90 : PTCLDY  
BUFFALO : 60/76 : SUNNY  
BURLINGTON UT : 63/80 : TSTRMS  
CHARLESTON WV : 62/87 : PTCLDY  
CHARLOTTE : 61/90 : PTCLDY  
CHICAGO : 60/82 : PTCLDY  
CLEVELAND : 64/76 : PTCLDY  
DALLAS FT WORTH : 72/94 : PTCLDY  
DENVER : 62/94 : PTCLDY  
DETROIT : 59/82 : PTCLDY  
GREAT FALLS : 65/100 : PTCLDY  
HARTFORD SPGFLD : 65/89 : PTCLDY  
HONOLULU : 75/89 : SUNNY  
HOUSTON INTCNL : 77/88 : TSTRMS  
KANSAS CITY : 68/92 : PTCLDY  
LAS VEGAS : 86/109 : URYHOT  
LOS ANGELES : 66/86 : PTCLDY  
MIAMI BEACH : 80/91 : PTCLDY  
MPLS ST PAUL : 62/82 : PTCLDY  
NEW ORLEANS : 75/84 : TSTRMS  
NEW YORK CITY : 70/91 : PTCLDY  
NORFOLK VA : 71/89 : MOCLDY  
OKLAHOMA CITY : 68/94 : PTCLDY  
ORLANDO : 75/92 : TSTRMS  
PHILADELPHIA : 70/91 : PTCLDY  
PHOENIX : 88/110 : SUNNY  
PITTSBURGH : 61/81 : PTCLDY
```

يبين الشكل التالي الموقع :National Weather Service

000
 FPUS12 KWNH 142244
 TAV12

TRAVELERS FORECAST TABLE...PART 2 OF 3
 NWS HYDROMETEOROLOGICAL PREDICTION CENTER CAMP SPR
 636 PM EDT SAT JUL 14 2007

TEMPERATURES INDICATE DAYTIME HIGH...NIGHTTIME LOW
 B INDICATES TEMPERATURES BELOW ZERO

CITY	FORECAST SUN....JUL 15		FORECAST MON....JUL 16	
	WEA	LO/HI	WEA	LO/HI
ALBANY NY	MOCLDY	65/83	PTCLDY	59/84
ANCHORAGE	MOCLDY	50/65	PTCLDY	52/67
ATLANTA	PTCLDY	71/86	PTCLDY	70/87
ATLANTIC CITY	PTCLDY	66/89	PTCLDY	66/90
BOSTON	PTCLDY	67/90	SUNNY	66/80
BUFFALO	SUNNY	60/76	PTCLDY	56/79
BURLINGTON VT	TSTRMS	63/80	PTCLDY	55/77
CHARLESTON WV	PTCLDY	62/87	PTCLDY	65/86
CHARLOTTE	PTCLDY	61/90	MOCLDY	69/88