Tangle: The creation of a trusted, task-based, distributed information system utilizing a peer-to-peer network

Jason Stuart Schneekloth
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
Schneekloth, Jason Stuart, "Tangle: The creation of a trusted, task-based, distributed information system utilizing a peer-to-peer network" (2005). Retrospective Theses and Dissertations. 20893. https://lib.dr.iastate.edu/rtd/20893

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Tangle: The creation of a trusted, task-based, distributed information system utilizing a peer-to-peer network

by

Jason Stuart Schneekloth

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Human Computer Interaction

Program of Study Committee:
Adrian Sannier, Major Professor
James Oliver
Tony Townsend

Iowa State University
Ames, Iowa
2005

Copyright © Jason Stuart Schneekloth, 2005. All rights reserved.
Graduate College
Iowa State University

This is to certify that the master's thesis of

Jason Stuart Schneekloth

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS**  iv

**ABSTRACT**  v

**CHAPTER 1: INTRODUCTION TO TANGLE**  1
- Motivation  1
- Defining Tangle  4
- Tangle Iterations  7

**CHAPTER 2: UNDERLYING NETWORK**  11
- WinP2P  14
- Security Concerns  20

**CHAPTER 3: TANGLE, THE APPLICATION**  23
- Displaying a Tangle  23
- Tangle Extension Interactions  32
- Security Concerns With Auto-Distribution  35
- Invitation Distribution  37

**CHAPTER 4: TANGLE EXTENSIONS**  42
- Creating a Tangle Extension  43
  - Step 1: Implementing ITangleExtension  43
  - Step 2: Filling in Required Data  44
  - Step 3: Receiving Events from Tangle and Talking to Tangle  49
- Undocumented Features  53

**CHAPTER 5: FUTURE WORK**  55

**APPENDIX: WINP2P**  60

**REFERENCES CITED**  73

**BIBLIOGRAPHY**  75
ACKNOWLEDGEMENTS

Above everything else I would like to thank my lovely wife, Claudia. Without your help, encouragement, strength, and love, working on this thesis would not have been an option for me. You have given me so much in life and you are where I draw so much strength from. You are the reason I am where I am today, and where I will be tomorrow. You believing in me is all I ever needed. Thank you, Love.

I would like to thank my Parents, Gene, and Jan, for sticking by me throughout my entire scholastic career. I know it has been a long time, but your love and support has made that time worthwhile.

I would also like to thank my committee members, Dr. Adrian Sannier, Dr. James Oliver, and Dr. Tony Townsend, for working with me and giving me guidance as to where to take Tangle. Giving a “real world” spin on Tangle greatly helped it’s development and I thank you all for that.

Finally, I would like to thank my major professor, Dr. Adrian Sannier for giving me a chance at VRAC. You gave me a lot of freedom when it came to everything I did here and that freedom allowed me to explore different solutions to Tangle. You were also there for me every time I had any kind of question and allowed me to tap into much of the knowledge you have learned throughout your career. You have been an invaluable resource and a true mentor throughout my time here at Iowa State. Thank You.
ABSTRACT

Every day more and more people from all walks of life use computers to communicate. Using a wide variety of new tools, including email, instant messaging, online chat rooms, and blogs, people of all ages have adopted computers as a means to keep in touch. Typically, each of these communication forms requires its own application, regardless of whom information is being sent to. For example, an email client is needed to send email to a friend, and a separate instant messaging client is needed to send instant messages to that same friend. Not only are different applications required to send information between the same two people, but the information is routed differently, using different network topologies and technologies, in order to complete what is essentially a simple form of communication between two people.

Email and instant messaging are just two of the many examples of computer mediated communication. Everyday it seems, people find another way to use computers to share information with one another. Many other examples exist, such as sharing pictures between friends, sharing music, sharing knowledge — the list goes on and on. These communications typically occur within groups — friends and family, a project team, a group or division within an enterprise. The information exchanged within these groups does not respect application boundaries. For example, the people a person shares pictures with are also likely to receive email and instant messages from that person.

This thesis discusses the design and creation of a comprehensive peer-to-peer communication system referred to as Tangle. The goal of Tangle is to provide a common framework and communication network to support communication in any form between a
group of people, providing a common entry point for any type of communication between groups of trusted peers. With this underlying infrastructure created, Tangle provides a base set of functionality which the everyday computer user would find most useful. Tangle also defines a mechanism to create new forms of communication within a group, called a Tangle Extension. These extensions provide unlimited extensibility to the set of task oriented communications possible between groups of peers using Tangle.
CHAPTER 1: INTRODUCTION TO TANGLE

Motivation

The motivation behind the creation of system like Tangle is rooted in the desire to integrate the separate applications and online identities one must maintain in order to perform the various communication tasks that define a group of people in an online environment. For example, consider a computer user who uses the internet on a daily basis to interact with a specific group of people. This user has an email address which he manages and can access via a distinct identity, a unique combination of username and password. This user has another identity for the instant messaging service he uses to communicate with the same people he emails. If this user also enjoys sharing pictures and creating photo albums to share with friends and family, this activity introduces a third identity to be managed in order to complete a different task with the same group of people he has been emailing and instant messaging. Perhaps this user would also like to create a storage area online, a shared file space allowing everyone in the group access to a collection of files regardless of the computer they are using -- another service for the group, another application and identity for our user to manage.

In this scenario the user must manage at least four separate identities for each service to interact in various ways with the same group of people. The interactions are media centric. The communication medium is the primary means of organization – the communication groups are implicit. Tangle reverses this organization to create a group centric model. In Tangle, the communication group is the primary means of organization, the communication modes are implicit.
This research emerged from work on the development of a trusted, group-based desktop search application [25]. This earlier work was focused on connecting a group of trusted peers together to create a common, shared, indexed file space, allowing search and retrieval from the shared space by group members. This shared private space utilized the same peer-to-peer file sharing mechanism that underlies anonymous file sharing systems like Kazaa and gnutella [19, 20]. Anonymous peer-to-peer file sharing creates huge shares of knowledge that are searchable by the world at large. Anonymous peer-to-peer has proven effective as a means to share well known content, such as popular music files, among essentially untrusted peers. Selection pressures tend to keep malicious and worthless content from proliferating, while valuable content finds a home on many of the machines in the peer to peer network. On the other hand, anonymous peer to peer networks are not effective at building databases of less well known content. In such networks, anonymous users have far less basis to filter through the vast amounts of information available to extract information that is personally useful.

Trusted peer-to-peer is an alternative to anonymous peer-to-peer. Unlike the anonymous case, in a trusted peer-to-peer network, the number of peers is small, and a trust relationship exists between the peers. A trusted peer-to-peer network is used to create a shared information repository that is smaller but denser than an anonymous store – smaller in that it holds far fewer bytes, but denser in that more of the information within the repository will be of use to the trusted peers within the group. Trusted peer-to-peer is designed to connect an identified group of people with a common goal in order to create a highly usable pool of shared information. We call these trusted peer-to-peer groups Tangles, to reflect the
idea that, because the information is being collected by a group with a purpose, the
information within the shared repository would naturally contain a complex net of
relationships not defined a priori.

During this work, we became interested in the idea that a single individual might be
interested in creating more than one trusted group, because people interact with different
groups of people in different ways. For example, a group of coworkers utilizing a file sharing
system would interact in very different ways than a group of family members might. During
the course of a day, an individual might interact with several different groups of people,
sharing distinct information for distinct purposes. The original system illustrated the
potential of trusted peer-to-peer as the basis for meaningful interaction between peers in a
trusted group, but it did not easily extend to manage multiple groups. Extending the concept
of a Tangle to multiple, distinct groups of peers became one of the principal research goals
for the work described in this thesis.

Our second principal goal also emerged as an extension of the original Tangle. The
original Tangle is file focused. Users share files of various types, the files are indexed, and
then Tangle supports text-based queries to retrieve sets of matching files. The file space is
assumed to be of diverse types. Modeled on internet search engine retrieval, the original
Tangle makes no assumptions about the information’s purpose.

However, if we imagine that a group of people is formed to share particular types of
information for particular purposes, then the opportunity exists to replace a generic file
retrieval mechanism with more specialized file interfaces, optimized for particular purposes.
For example, if a user were sharing pictures of a vacation with family members, one would
want that information displayed in a completely different manner than if that same user, in
the same group of family members, was also maintaining a kind of family blog space. In
each case, the information presentation is as important as the information itself. Extending a
generalized interface to support all forms and purposes of sharing is impractical and
undesirable. *Thus the second research goal for this work was to develop a framework for
creating “sharing skins” – custom, task oriented interfaces that provide access to a trusted
peer-to-peer file repository.*

**Defining Tangle**

Given the two foregoing goals – to extend trusted peer-to-peer sharing in order to
support multiple groups and provide a framework for creating task oriented skins, one can
begin to outline a solution. The version of Tangle described in this thesis aims to provide a
generalized solution to sharing information between groups of trusted peers and to provide an
infrastructure to display this information in a task optimized way. The target audience of
Tangle is the non-expert computer user who is part of an online community, who uses email,
instant messaging and other forms of information exchange on a regular basis to
communicate with distinct groups of colleagues, friends and family.

The groups of people Tangle is designed to support are not meant to be large,
sweeping, all encompassing groups of unknown peers. Instead, Tangles bring together small,
tight-knit groups of trusted peers, allowing them to share task directed, useful information
with the other people in those groups. This is an important distinction between Tangle and
anonymous peer-to-peer solutions.
There are two main pieces of Tangle which must be considered when defining it, based on how people interact with it. There are two main interaction mechanisms, each of which targets a different audience. These interfaces are the user’s interface and the developer’s interface. The user’s interface makes Tangle’s functionality accessible, allowing the user to control the overall information flow within a Tangle and interact with the other members of the group. The developer’s interface is targeted at builders and developers of Tangle Extensions. This is the interface that allows a developer to build on the underlying Tangle platform to create a task optimized skin for the user. The developer’s interface is a critical element in extensible software. Given that Tangle is designed to allow customizable display of any type of information (ranging from simple image sharing, to a fully developed group-based product development management), the developer’s interface must make it easy for developers to create information skins and controls for Tangle. The developer’s interface is known as the Tangle Extension (TE) system.

Another issue addressed in this thesis is the so called “out-of-box” experience - the initial experience which a user has with Tangle. Since Tangle is unlike any system which an average computer user would currently be familiar with, there are no natural analogs for a user to fall back on. Exploratory learning can be helped along if initial base judgments on a new application are leveraged by an already known concept [23, 24]. The known concept provides a familiar jumping off place for the user, promoting exploration within the new application. At some point in the development of a technology, a broad community of users builds an expectation around a technology platform such as email or instant messaging. Users share the expectation that an established technology platform will provide a base set of
functionality. Once this level of ubiquity is reached, a platform’s potential can be fully explored. Perhaps this will occur for trusted peer-to-peer systems at some point in the future, but at this point a system like Tangle establishes few expectations in the mind of a new user.

In this sense it is what Moore [21] would call a disruptive technology, one user must make significant behavioral changes in order to adopt. The first step in a disruptive technologies journey to ubiquity is determined by the initial experiences of inexperienced users. One of the principal challenges in introducing a disruptive technology is to make it seem familiar. A way must be found to relate Tangle’s capabilities to something the user is already familiar with. If the two tasks, one new and one familiar, can be related in a natural and coherent manner, then the adoption of the new technology is encouraged. The initial user experience designed for Tangle will be drawn out in greater detail later in the thesis.

Figure 1. The different components which work together in order to create the Tangle System. The arrows show the interactions between the different components with the Tangle Application acting as an all knowing intermediary.
Tangle is made up of four main parts. The four parts are the peer-to-peer (P2P) networking layer, the Tangle Application, the user experience, and the developer interface. Figure 1 shows the different parts of Tangle and how they interact with one-another.

Each of the four parts has specific requirements which must be met. These requirements are the foundation for Tangle’s architecture and are based on assumptions made during the design phase of Tangle’s creation. The Tangle design described in this thesis is the result of an in depth process of testing different paradigms of grouping and peer interaction. The current architecture arises out of three distinct iterations.

Tangle Iterations

Tangle has gone through three distinct design iterations before arriving at the incarnation described here. The first iteration of Tangle has been briefly talked about already, a kind of extended desktop search engine that combines the digital assets of a trusted group of people into a common searchable space. This first iteration proved a valuable exercise in defining many of the available group functionalities and interactions which eventually made their way to this version. As we pointed out above, the first version of Tangle assumed a user was a member of only one file sharing group and it provided only a generic file-based user interface.

The second Tangle iteration focused on changing the underlying Tangle framework to facilitate the creation of task specific sharing skins. Unlike the original Tangle’s generic file based interface, these skins, running as separate applications, would provide task optimized interfaces to file collections created by a group of people for a specific purpose. Tangle, in
this iteration, served as an always-on peer-to-peer file exchange service which would broker
calls between these external skins and the underlying peer-to-peer networking functionality
presented to those applications. The skins were responsible for creating groups of peers
through the Tangle interface, and these groups were application dependant. This design
allowed an individual to be a member of multiple groups; however each group was defined
behind a particular skin, organizing a particular type of information. A single group of peers
did not naturally span more than one skin.

Figure 2: Application design for the second Iteration of Tangle. Tangle Manager acts as the shared
service by which separate and distinct Tangle Types could connect to in order to create and maintain
groups of peers.

In this iteration, an object called the Tangle Manager exposed the peer-to-peer
functionality to the separate and distinct skins, or Tangle Types. Figure 2 shows a diagram of
the different parts of Tangle in this second iteration. In Figure 2 one can also see how the P2P functionality (WinP2P in the figure) was separated from both the Tangle Application and the skin. This allowed the Tangle Application to control what P2P functionality was exposed to the developer.

This iteration supported the construction of independent applications. Using this model, integrating group interaction functionality into existing applications, such as a photo album, was straightforward. The photo album software itself – all of the image editing and management capabilities – are easily separable from the tasks which Tangle would perform to create groups of peers which would be sharing photos with one-another. In this approach, the photo album software would be responsible for creating the groups (through Tangle), maintaining the groups, and adding data to the groups. Multiple groups of peers could be managed through this single application, although the complexity associated with such a task is considerably greater than that of only allowing interaction with a single group.

The complexity in managing more than one group is one of the principal drawbacks of the second iteration. Because control for group definition lies within the application, Tangle does very little to limit or censor what can and cannot be done. This provides a great deal of flexibility for the application developer, but at the expense of increased complexity.

A second drawback to the second design iteration lies in the fact that groups of peers could not naturally span applications. If a skin for Tangle Type A created a group of peers, that group was unknown to the skin for Tangle Type B. If that skin provided functionality useful to the group, Tangle Type B would be responsible for recreating and managing that group of peers again, separately from Tangle Type A. This presents a problem if a particular
group of people wants to add additional skins, to share additional kinds of data. Instead of a single group, each of these skins must maintain the group separately. In situations where the group is primary, the second design iteration fell short. This led to a group-centric viewpoint, where Tangle facilitated the definition of persistent groups to which skins could be added to facilitate task specific access to file collections. In this formulation, the groups of people are the organizing principle, not the task specific applications. In this model, a group of peers shares a variety of different types of data, with a variety of different skins available to them to interact with that data, rather than building the group around a particular skin.

Thus the incarnation of Tangle described here was developed. This version of Tangle places priority on defining groups of peers and no base functionality exists for interacting with those groups without a skin, or Tangle Extension. These skins act as plug-ins to the system and add new and unique interactions between groups of peers. A single group of peers can have access to many different skins; in fact, all extensions loaded into the system are available to all groups of peers. This promotes Tangle’s extensibility while allowing the user to concentrate on the creating and managing the groups of people he or she knows.
CHAPTER 2: UNDERLYING NETWORK

The underlying peer-to-peer (P2P) network plays a critical role in the overall design of Tangle. Ultimately the functionality present in such a network will define what can and cannot be done by the system and will also be a limiting factor in how Tangle will ultimately be perceived by the users [1, 2]. Regardless of the network topology, one of the most important concerns to the end user will be the overall responsiveness of the system and the noticeable lag the P2P network will have when performing tasks. If the Tangle user has to wait for extended periods of time between submitting information or data to the group of peers and receiving a response from the group, this will make the user feel unconnected, diminishing user enthusiasm.

On a very basic level, a P2P network has many advantages over client-server networking. Not only can content and resources be shared from the entire network, but P2P networks scale easily and are more reliable and fault tolerant than standard client-server approaches [4].

There are many different protocols and network types which could be used to implement Tangle. The choice to use a P2P network was made for many reasons. One of the main reasons for choosing a P2P network is that knowledge stores created on an individual’s computer are private to that particular user of Tangle and/or the group he or she is a part of. P2P topologies eliminate intervening servers, restricting the information within the knowledge store to trusted nodes on the network, rather than passing the information through a server between clients. The P2P topology also leads to a reduction in attack points for malicious activity and decentralizes the system, decreasing its vulnerability to attack or
failure. Attack points are reduced based on the fact that there is no centralized server which can be attacked in order to cause disruption in Tangle.

Other reasons for choosing a P2P infrastructure lie in our desire to make the system autonomous, or self-organizing, in nature. The purpose of Tangle is to provide an easy medium for sharing knowledge and performing tasks within groups of peers, and to minimize the infrastructure required as a prerequisite to sharing that knowledge. P2P has a distinct advantage over client-server architectures in this regard. Finally, P2P is more suited for Tangle because with P2P the system scales easily with group size and does not rely on server hardware defining what a user can and cannot do.

There are concerns with a P2P network that must be addressed, mainly dealing with performance issues when compared to traditional client-server architectures. The main performance bottlenecks arising in P2P networks are bottlenecks associated with the individual nodes in the network and their connections to the internet. As consumer computing power and connection speeds continue to increase, these issues become less and less significant. Certainly for many user groups, the combination of desktop power and connection speeds are sufficient to provide excellent performance for a wide range of networking tasks.

It is assumed that the tasks performed by the network are to be transparent to the user, regardless of the network architecture. Of course some tasks are most efficiently performed in one architecture versus another. For example, within a client-server architecture, a query made to the system by one person would be easily propagated to others logged into the system and then the results returned in a timely fashion. This is a more difficult task within a
P2P environment for many reasons including, but not limited to, network latency and individual node performance issues. Other issues which arise have to do with proposing an actual search to a group of peers within a P2P network, information retrieval mechanisms, and security concerns which the user may have about what and how he/she can limit access to his/her information, or knowledge store. More on this subject will be discussed in the future work section of this thesis.

For the initial iteration of Tangle, Jabber [5], an open source P2P network was chosen to provide the communication backbone. Jabber is a P2P networking protocol originally developed to support instant messaging (IM) tasks. Jabber provides a basis for creating and maintaining a single group of peers with which a user can exchange messages. Most of the messaging in this iteration occurred in true point to point fashion, between two distinct peers. Very few messages were sent to all the members of a peer group, other than full group queries, and the system was designed to exploit this fact. The IM protocol was very useful because direct messaging was a part of that protocol.

In the second and third iterations of Tangle, the Jabber protocol would no longer serve as a networking protocol, because these new iterations were based on the idea of having multiple groups of peers to interact with. Jabber could not easily account for this type of interaction. A new protocol needed to be identified to meet the design objectives of the second and third iterations.

In the end, Jabber was replaced by Windows Peer-to-Peer Networking. The Windows Peer-to-Peer Networking (WP2PN) platform is a P2P networking platform that is secure,
scalable, distributed, serverless, robust with respect to failure and attack, self-tuning, and self-repairing.

**WinP2P**

The raw form of WP2PN is a complex system written in a lower level programming language (C) rather than the target language of Tangle (C#). The programming language decision rested solely on the fact that a minimal number of developers (one) were available to actually code Tangle. The WP2PN API itself is written in C and therefore takes a very functional approach toward programming and is in no way object oriented. Since the rest of Tangle was developed using an object oriented approach, the WP2PN API was wrapped in a higher level abstraction in order to be usable by Tangle and facilitate the rapid application development available in C#. The wrapping procedure uses platform invoke calls, a convention defined within the C# language to allow interoperability between C# and unmanaged code [6, 7].

The components which make up the WP2PN platform consist of the following:

- **Graphing** – This component is responsible for maintaining connectivity between nodes which is referred to as a graph. This part of the platform is also responsible for data replication across the graph.

- **Grouping** – This component defines the security model behind group creation, invitation, and connection to a peer group.

- **PNRP** – This component provides peer name resolution and is referred to as the Peer Name Resolution Protocol (PNRP).
- **NSP** – This component, the Name Service Provider (NSP), provides access to the PNRP component.

- **Identity Manager** – Enables the creation and management of peer identities.

- **Microsoft TCP/IP version 6 protocol** – Provides the underlying network transport mechanism.

Tangle interfaces primarily with two of the six components of the WP2PN platform. Figure 3 shows the architecture of WP2PN and the two components Tangle interfaces with: the Identity Manager and the Grouping component. The other components of the API provide access to lower levels of the platform. The Identity Manager and Grouping component provide higher level abstractions of these lower level functions. The abstractions are sufficient for Tangle, thus the WP2PN wrapper, which we call *WinP2P*, covered only the functions from these two components within the API.

![Figure 3: Windows Peer-to-Peer Networking architecture, taken from, http://www.microsoft.com/technet/prodtechnol/winxppro/deploy/p2pintro.mspx](http://www.microsoft.com/technet/prodtechnol/winxppro/deploy/p2pintro.mspx)
In using WP2PN, the two most important considerations are how a user of the system is defined, and given a set of users, how are those users connected together in some fashion. The WinP2P wrapper’s main namespace, called WinP2P, contains two sub-namespaces referred to as WinP2P.Identity and WinP2P.Grouping. The sub-namespaces account for the identity functionality and grouping functionality respectively. These sub-namespaces provide a developer with access to the minimal amounts of functionality necessary to create groups of peers, manage those groups of peers, and share data across those groups of peers.

The first structure which one has to deal with is the idea of wrapping what an identity is within the system. Within WP2PN there exists no defined structure that holds data for an identity, but rather an identity within the system consists of a Unicode string, or peer name, which is unique for every user and a distinct grouping of functions which deal with the management of this identity string. There is also a human readable form of the peer name, which is referred to as its friendly name. The friendly name is held separately from the Unicode string representation.

Two sets of actions are defined on an identity; the actions which the identity can perform and the actions performed on the identity. To better distinguish these two sets of actions, it is useful to point out that the identity controls the groups that the identity is a member of. Therefore, it is natural for the identity to have functionality which creates a new group, to remove one’s self from a group, and to obtain a listing of groups for which the identity is a member of. The functionality just described provides the functions which are available within an identity. The objectifying of an identity also contains the properties for
retrieving the peer name and the friendly name for the identity. We will define exactly what a group is and how one interacts with a group of peers soon.

The next piece of wrapped functionality, after the definition of an identity, is the mechanism for creating and managing identities on a given system. In WinP2P, this is referred to as the \textit{P2PManager}. The most important functions available to the developer from within the P2PManager are the abilities to create an identity, delete an identity, and retrieve the set of identities defined on the system. There is a second portion of the P2PManager which must be discussed as well. In order to use WP2PN, one must make function calls to \textit{startup} and \textit{shutdown} the system. This functionality is also exposed from within the P2PManager, giving a unified entrance into WinP2P.

The P2PManager itself is based on a singleton design pattern so that multiple applications using WinP2P can run concurrently without incident. Data shared between concurrent applications, within the space for WinP2P, can be specifically managed. This does not come into play, however, since there is no data to manage as all tasks are performed at a lower level within the WP2PN platform itself. This approach allows for future extension of WinP2P into areas of the WP2PN which are not specifically managed and would need to be managed by WinP2P.

With an identity defined for a user, the WinP2P wrapper must now provide the mechanism for collecting this identity together with other identities to form a \textit{peer group}. Similar to an identity, a peer group in WinP2P is a concept which performs actions itself, such as creating invitations for people to join the peer group, and is also an abstract idea which has actions performed upon it, such as creating and/or removing a peer group. The
peer group encapsulation itself consists of the most basic functionality of maintaining the group of peers: inviting, connecting and disconnecting. Inviting new identities (peers) to the group is the process of creating an invitation to the group and handling that invitation once it has been received. Connecting to and disconnecting from the group are also pieces of this abstraction.

Data sharing within a peer group in WP2PN is based on the concept of a peer record. In WP2PN a peer record is a generic container for data that is spread across an entire peer group. When a new record is added to the group, it is automatically propagated to all members of that peer group. Without yet defining what an actual record consists of, the WinP2P wrapper must provide a mechanism for members of peer groups to add, modify, and delete records in the peer group.

Using a record, data must be passed to the entire group, or it is not passed at all. There is no way to pass data to only one member of the peer group using a peer record. This would be a useful feature for many reasons, a simple one being returning a query result to a peer who queried the group. Since the result is only important to the peer asking the question, the reply should only be sent to that peer. To address this need, WinP2P defines a mechanism to create direct connections between two members of a peer group.

Finally, a peer group in WinP2P contains certain properties, similar to ones defined for identity. Each peer group contains a peer name (unique Unicode string), a friendly name, and a string representing a comment about the peer group. The comment provides a short explanation of group purpose, or anything else the creator of a peer group would like to include.
With definitions of a peer group and the identities within that peer group, we can now define the main source of interaction between those identities in the group. A record, or peer record, acts as a generic bucket for distributing data across a peer group. The data held within a peer record is a collection of bytes, specified by a pointer in memory and the number of bytes. WinP2P wraps a record as an array of bytes. Any data or object within the .NET framework can be serialized into an array of bytes, so the WinP2P record allows any data type to be serialized, passed to the group of peers, received by an identity in the group, deserialized, and then handled accordingly by everyone in the group.

Peer records are strongly typed. Each peer record has a property referred to as type. This property, within code, is of type System.Guid (Globally Unique Identifier), and is handled on an application level. For example, an application creating a new peer record would add a specific type of data to that record, give it a System.Guid for that type of data and add the record to the peer group. This allows all other applications the ability to process the data in the record in the correct, type specific manner. Peer records also contain an expiration time. This expiration time indicates the time at which the information in the record is no longer valid. A record remains on the peer group until its expiration time passes. So, for example, if members A, B, C are all online and connected and member A adds a record to the group, it is propagated to B and C. When member D connects at a later time, the record is automatically propagated to D as long as the expiration time on the original record is later than the time D connects to the group. This propagation is handled directly by WP2PN. Lastly, the peer record contains information about the identity that created the record and the identities of any modifiers of the record.
WinP2P defines a direct connection mechanism which can be used to create a connection between any two identities within a peer group. This direct connection provides a bridge for streaming data between exactly two identities as efficiently as possible. Once a direct connection has been established, that connection can be used to send and receive data with another peer. The send and receive processes can happen synchronously or asynchronously. The direct connection abstraction also contains information about the identities of the endpoints of the connection, along with a unique id associated with the particular direct connection instance. Direct connections must be closed explicitly. This cleans up any resources associated with the process of sending and receiving data directly with another peer.

A more detailed explanation of WinP2P can be found in Appendix A toward the end of this thesis. In this appendix, all of the functionality exposed within WinP2P is detailed and explained.

Security Concerns

One of the principal attack points in a system like Tangle is the underlying networking layer. It is the networking layer that ultimately provides remote access to Tangle and all of it’s functionality. By their very nature, peer-to-peer networks are somewhat more difficult to attack and bring down than their client-server based network cousins. In client server architectures there is a single, known, and often times public attack point – the server – which increases their vulnerability. P2P architectures on the other hand have no central point of attack, save perhaps the seed servers needed to seed entry into the network. Compared to
servers in a client-server topology, seed servers are quite simple. A seed server is only responsible for getting new nodes into an already existent P2P network. Unlike the server in a client-server architecture, the seed server is not responsible for data transmission between nodes, and there can be many seed servers within a completely distributed P2P network. Thus, a main concern for seed servers is to be resilient to denial of service (DoS) attacks. In fully distributed P2P networks, DoS attacks are very difficult to perform since all nodes within the network are disconnected and redundancy is a built in factor of the network. There exist other weaknesses based on the network topology, such as authentication if that is done via the seed server, however, as will be stated later, our P2P topology does not do authentication at this point, but rather relies on nodes within the network for authentication. Dealing with DoS attacks has become commonplace on the internet today, but it should be pointed out that a successful DoS attack on a seed server does not bring down Tangle in any fashion; Tangle’s P2P network can maintain connectivity with or without a seed server.

WP2PN has been designed to work well in a serverless environment – it works even better if servers are in place. One of the largest concerns comes into play with attacks which are similar to what is referred to as a Sybil attack [3]. The Sybil attack is based on an assumption that an attacker can take over, or control, a large number of nodes within the overall P2P system. In the case of WP2PN, an attacker would have to control a large number of nodes in the P2P graph of nodes. This attack, however, is extremely difficult and highly unlikely to occur, although not impossible because of the nature of WP2PN [3, 4]. The attack cannot be thwarted without a centralized trusted agency to certify identities. It is possible distributed trusted agency, so to speak, which is used by WP2PN, can be attacked,
although this attack is extremely difficult to accomplish because of the shear number of
nodes which must be compromised within the graph [3]. The grouping component of
WP2PN is used as a security provider for the graphing component.

A formal security study on WinP2P itself has not been performed. When objectifying
WP2PN in order to create WinP2P, certain assumptions came into play and certain shortcuts
had to be performed for the sake of time. Clearly WinP2P can be no more secure than
WP2PN; in its current form it is demonstrably less secure.

Unlike WP2PN, WinP2P does not provide mechanisms to encrypt the data being
exchanged between peers. Although automatic encryption mechanisms are supported from
within WP2PN, those mechanisms were not abstracted to WinP2P, since WinP2P utilizes the
grouping component. The grouping component of WP2PN is a secure mechanism for
managing credentials associated with identities and publishing records within a peer group,
but it does not encrypt the data exchanged between peers. WinP2P leaves the task encrypting
of the data held within a record up to the application. This allows for greater control over
how the data is protected and sent over the internet, but it is a significant burden on the
application developer.

The process of securing information on the application level is a subject which will
not be mentioned in this thesis and a solution will not be proposed. Tangle, itself, does
nothing to protect the data being stored in records on a peer group at this time. The
implementation of a secure application which communicates with, essentially, unknown and
initially unverifiable peers is a subject beyond that of the initial goals set out for Tangle.
CHAPTER 3: TANGLE, THE APPLICATION

Tangle is a relatively complex piece of software. As such, one of the jobs of the user interface is to manage that complexity and minimize the degree to which users are aware of "how" the functionality is being provided. So the software must do a lot behind the scenes to make the user experience as simple as possible. This chapter focuses on the design decisions that control the user’s initial experience with Tangle. We also discuss the underlying technology that provides the user’s experience.

Displaying a Tangle

Conveying the concept of a shared, interrelated but non-hierarchically organized data store to users is a central design issue for Tangle. The first iteration accomplished this task by presenting the idea in raw form. Users perceive a Tangle as an integrated set of shared folders, found on the computers of multiple friends, that are indexed as a single unit, and that can be searched as a single unit.

The second iteration enhanced the original concept by introducing two additional layers of complexity: Tangles in the second iteration were type-specific, with type-specific interfaces, and users could create multiple, distinct type-specific Tangles. The presentation or display of these type-specific collections was left to the Tangle application developer. Tangle in the second iteration provided underlying functionality, but each type-specific Tangle was free to depict the shared information through whatever interface the developer chose to implement.
The third and current iteration of Tangle simplified the previous iteration by making the creation of groups of peers the central organizing construct. Any group might share multiple, type-specific repositories, but Tangle provides a consistent way for users to define groups and create distributed, type-specific repositories for those groups.

We tried two different user metaphors for presenting this group structure to users. The initial attempt presented the Tangle application to users directly, as a kind of Tangle administration app, as an application to create, manage and access groups of peers. The Tangle application was built around a container form which housed all of the defined peer groups. Using the container form, a user could review the various groups of peers he was a member of, define and manage new groups of peers, and invite or remove members from existing peer groups. Further, a user would specify what type-specific interfaces would be exposed to individual groups. Figure 4 shows this first attempt at creating the groups of peers held within some type of container form which would also be responsible for management of the individual Tangles.

The container form which housed all of the individual Tangles for the system was a good test bed to define functionality which needed to be present within the application. During ad hoc user testing however, it became clear that most of the functionality presented on the left side of the form in Figure 4 was rarely used after initial system setup. The most frequent user functionality – sharing files, making queries, accessing results – occurred through the right click menu on Tangles themselves. During ordinary use, the container form was a “click through” that used large sections of screen area to present little used options.
Based on these initial tests, we redesigned the Tangle presentation to push the common Tangle administration functions into the background after the initial setup. To simplify the user’s experience we developed an interface that presents individual Tangles (defined as groups of people sharing particular files through particular skins) as standalone objects. Peer groups appear on the users desktop as individual, oversized, icons which remain at specific locations on the user’s desktop. Right clicking on these icons allows users to interact with these groups, and configure them.

Figure 5 illustrates this concept with two different Tangles, *VRAC* and *Temporal Space*. Configuration and status options, common to the infrastructure, underlying all the
groups are relegated to an icon in the system tray. The right-click menu for this system tray icon can also be seen.

Figure 5: Two Tangles, VRAC and Temporal Space, shown on the desktop.

The separation of the peer groups from the application which manages them is an unusual approach for an application program. Acquainting users with Tangle's mode of operation must therefore be undertaken with particular care. The task of introducing the user to this organization construct is performed by an initial wizard that guides the user through the process of creating an initial Tangle.

Making individual Tangles the primary focus of the interface allows a user to become acquainted with Tangle gradually, beginning with one group and expanding his/her focus gradually as they become more familiar with the idea of a managed group of peers. To further simplify the concept, a wizard can be focused on setting up a single group of peers for
a single, well defined purpose. Focusing on providing a particular type of sharing to a single group of users allows novices to experience Tangle’s application benefits as quickly as possible, with the smallest learning curve. Later, as users grow familiar with the operation of this first group, they can expand to more than one group, or to other type-specific sharing areas.

This wizard is run when the user starts Tangle for the first time and walks the user through five steps in creating a Tangle with a given purpose. The type-specific application focus we chose for this first wizard is photo sharing. Photo sharing is ideal for bringing novice users into contact with Tangle because there are already so many natural metaphors. Most people are familiar with exchanging photos among a circle of family and friends in the physical world. In recent years, with the rise of digital photography, people have extended this physical action to the digital domain. But obstacles exist. Photos are often large files, making them awkward to exchange by mail. Web based digital photo galleries are possible, but configuration of these galleries requires expertise, and many people are reluctant to share photos outside of their trusted circle of friends and family.

Tangle provides an excellent base for creating a photo sharing circle. The circle forms the peer group, the photos the type specific data they exchange. Later, a user may decide to create more circles, or may want to expand the kinds of information he/she exchanges with a particular circle.

To illustrate the power of Tangle, we created tFoto, a Tangle based service to accomplish the same goal that hosted photo album solutions provide, but within the serverless environment that is Tangle. A user’s tFoto experience is based on natural
metaphors, but provides an excellent entrée into Tangle’s more complex features. A user is attracted to Tangle to share photos. tChat is bundled with tFoto, so users, after mastering tFoto’s features, can explore extending other forms of information exchange among the members of the existing group.

The wizard has two parts: one for the founder of the circle of friends, the other for the novice members the founder invites to join. The founder’s wizard prompts the founder to identify by email address a group of peers he wants to share photos between. The wizard installs Tangle, and then creates a tFoto enabled Tangle and issues email invitations to the group of peers identified by the founder. The wizard then prompts the founder for an initial set of photos he/she would like to share with the newly created group.

The wizard then starts Tangle, which displays the newly created photo sharing group as an oversized icon on the desktop. An html documentation page is opened within the users default web browser to inform the user what has just happened and how to access the tFoto gallery from the newly created group. With this information, the user can repeat these steps once he/she has learned where the Tangle is located. This initial experience gives a task to the user so that he/she is able to give purpose to Tangle and to provide motivation for further exploration of the application.

Each of the invited members receives an email from Tangle inviting them to join the tFoto circle created by the founder. Instructions for expert users are included, but the focus is on novices. Novices are instructed to download and run a custom wizard that sets about the job of linking them to the founder’s circle. Once they have accepted, and selected an optional initial set of photos to share, their answer is sent back to the founder for acceptance. Tangle
creates the icon on the joining members desktop and the circle is complete. Once a member, the invitee is a peer and can share photos, view shared photos, and invite other members.

The Tangle icon not only serves as a placeholder for the Tangle itself, but also serves as a status monitor which a user can quickly view in order to get information about the different happenings within the Tangle without opening the different extensions which are available to the Tangle. Along the bottom of the icon’s area resides a status area which the Tangle uses to inform the user of the online/offline status of the Tangle and where the tangle extensions can inform the user that there is new information within the group dealing with that particular extension which should be handled by the user. Figure 6 shows the notification area with the tChat notification present for the Tangle, VRAC.

![Figure 6: tChat's icon overlay displayed just to the left of the online notification overlay toward the bottom right of the VRAC Tangle.](image)

This notification area provides quick access to status changes, e.g. knowing that according to the tChat overlay, there are messages which the user has not seen inside tChat. The user can then open the tChat extension and view the messages he/she has not seen from
the group. The online/offline status notification area of the Tangle has three states, online, offline, and connecting, which can be seen in Figure 7. These three states inform the user of the current workable state of the Tangle. When the Tangle is in the offline or connecting state, no notifications from the group are sent or received. Only when the state of the Tangle is in the online status can interaction with the group occur.

![Figure 7: The different notifications of online status given to the user. From left to right, they correspond to offline, connecting, and connected.](image)

The individual Tangle icons are representations of the peer group and their right-click menu allows users to interact with those groups. This right-click menu can be broken up into three main areas, the extension area, the interaction area, and the miscellaneous area. Figure 8, below, shows this right click menu being used.

The first menu section is devoted to the tangle extensions, the skins, which are currently registered with the system. The skins provide type specific access to sharing and accessing particular types of information that have been shared with the group. The skin menus each “walk” to their own secondary menu. This menu, defined by the extension
developer, provides access to an extension’s presentation and configuration interfaces, and will vary from skin to skin.

![Figure 8: Tangle icon right click menu with two extension, tChat and tFoto, installed with the system.](image)

The second menu section provides access to features common to all Tangles. These functions are core to creating and managing the peer group, apart from what information is being exchanged. Connecting to and disconnecting from the peer group are the first two items in this menu. These are followed by less frequently used items, such as inviting a new peer, accessing the configuration properties or viewing the list of peers in the group.

The final section of the right-click menu presents features that are rarely used and do not fit in the other two groups. Two functions reside in this area, locking the desktop, and leaving the Tangle entirely. Locking the icon to the desktop makes it immovable. Leaving a Tangle withdraws the user from the peer group entirely. There is a confirmation message which verifies the user wishes to leave the Tangle to avoid accidents.
Tangle Extension Interactions

The underlying Tangle application provides a method for creating skins – custom interfaces that allow a user to share and access type-specific content within a peer group. The developer’s interface defines three principal interfaces that a Tangle Extension must support.

The first interface allows the extension to be visible through the Properties menu. The properties interface allows an extension to register itself with Tangle (allowing it to show up in the list of possible extensions) or to unregister itself. This interface provides a stylized mechanism for extensions to give information about the service they provide. Figure 9 shows how this interface is used to gather information about two registered extensions, tChat and tFoto. As can be seen in the figure, registration of a new extension and removal of an old

Figure 9: The extension tFoto being displayed in the extension management portion of the user interface for Tangle.
extension can be expressly performed by the user of Tangle. Along with this expressly defined functionality, one can also get an explanation of what the selected extension’s purpose is. For example, in Figure 9, we can see that tFoto is, “A simple photo album application which allows for groups of peers to create and maintain albums of digital photographs with others.” There is also space for a URL which points to a website where the user can get more information about the extension which is registered with the system. This consolidated space provides the user with one access point when dealing with Tangle Extensions.

A group of peers can only interact with one-another if they all share a common base of functionality for interaction. For example, let us look at a group of two users, A and B. Currently starting out, A and B each both have the same extensions and they both have the newest, most current versions of the extensions. Let us now suppose that A goes to a website to update his tChat extension to the newest version and person B does not. Next, A and B make a connection with each other via Tangle, through their common group, and B’s Tangle detects that he does not have the same version of the tChat extension that A has and that A has a newer version of the extension. Tangle then informs person B about this version mismatch and asks B if he would like Tangle to automatically update the extension. Person B says yes, indicating that he does want the most up to date version of the extension. Therefore as a result, person B’s Tangle contacts person A’s Tangle and brokers the reception of the updated extension which person A holds. When the update is done, Tangle is then restarted for person B. This process allows both A and B to keep their extensions updated.
and only one person was required to actually download the updated version of the extension from a website.

This type of interaction exchange also works for extensions which one member of the group has, but others do not. For example, if someone in a group downloaded and registered a new extension with their Tangle, all the members of all the groups that person is a part of will then have the ability to retrieve the new extension, via Tangle, so that interaction can occur using that new extension. This type of self-propagation of both new extensions and updated extensions helps to keep the entire system up to date. Figure 10 shows this idea of brokered information.

![Brokered information](image)

Figure 10: Brokered information made between two groups of peers who have no knowledge of one another but share a common member.

In the figure, group member C is a member of two Tangles, or groups of peers, and group member A, in Tangle Group 1, is the person who, for example, registers a new extension which nobody in Tangle Group 1 or Tangle Group 2 has. Member C retrieves the
new extension from A the next time they are both online within the group. Since C is also a
member of Tangle Group 2, this notification of a new extension is made to Tangle Group 2.
Everyone in Tangle Group 2 can then retrieve the extension from C. The two groups have
nobody in common except user C. Eventually, this process will lead to a near 100% distribution of anything dealing with new or updated Tangle Extensions across the network created by Tangle users. This type of auto-distribution functionality can also be used to distribute updates to Tangle itself, so that everyone using the software can seamlessly update their systems on a need-to-update basis, permitting everyone within the group to remain current with everyone else in their group when it comes to Tangle, the application, and all of the extensions used by Tangle.

Security Concerns With Auto-Distribution

Given this automatic update procedure, there are major security concerns which must be addressed, and are not currently addressed within Tangle. These concerns reside in the fact that Tangle does no verification of Tangle Extensions. If an outbreak were to exploit the auto-distribution functionality of Tangle, there would be no way to stop it. There has been some research done to counter such attacks, by Vlachose, Androuthsellis-Theotokis, and Spinellis. They propose the design of a P2P system which has the following characteristics:

[9]

1. Ability to detect if a virus or worm is propagating through the network
2. Automatically dispatch warnings to other peers within the group
3. Take specific precautions to protect their host by hardening security during and epidemic

The system’s goal is to detect and minimize the impact of malicious programs which could easily propagate via Tangle. As with any system, its overall usefulness and power will be limited by the effectiveness of its detection mechanism. The research proposes simply monitoring the log file from a separate, or third party, intrusion detection system to signify when an attack may be occurring to a given node in the P2P network. In the example of Tangle, periodically, one could verify the hash of the extensions on the system with an address which was initially given by the extension when it was accepted as valid and installed per the user request.

For example, when a user installs a new extension, he or she is accepting that it is, in fact, a valid extension and a website address can be received via that extension which contains the hash of the particular version, this hash can be recomputed in order to verify the extension is in fact what the user is wanting to install. This is just one way in providing a secure mechanism for verifying valid tangle extensions and protecting the users of Tangle. A malicious user could install an extension which he/she knew was malicious in order to propagate it through the system. Attacks like this could be devastating, especially when a perceived friend is the one who registers the new virus-like extension for the group. As a last resort to such propagation, interaction between Tangle and antivirus software should be used in a similar fashion by which antivirus software interacts with today’s email clients. This provides an end barrier which would help to stop the propagation of malicious code through the users of Tangle.
Invitation Distribution

The final piece of the Tangle application which needs to be discussed is the process of adding new members to a Tangle one is currently a member of. The underlying network requires an invitation to be made in order to invite a user to a group of peers. In order to create an invitation, two things are needed: the inviter’s identity, and the invitee’s identity. The inviter’s identity is easy to obtain since he/she is the person who is creating the invitation for another. The interesting piece of the problem comes into play when obtaining the invitee’s identity. If the invitee is already a member of another group which the inviter is a member of, then the underlying network has no problem obtaining this information. If the invitee is an unknown user, meaning he/she is not a member of a group that the inviter is a member of, then the problem of retrieving the identity from the invitee becomes apparent. Tangle, or rather the underlying networking requires a Unicode string which represents one’s identity in order to create an invitation to a peer group. The invitation distribution process hinges on being able to request and retrieve said identity. This problem has been solved by assuming the users of Tangle already have one communication mechanism in common with everyone else who will use Tangle. This communication mechanism which we assume to be present is e-mail. E-mail is a relatively slow communication mechanism when compared to a direct connection between two computers on the internet, (which is how fast the invitation process could occur without the identity distribution problem) but it allows for Tangle to remain distributed and not require a central server. By using e-mail, Tangle can connect two people who know each other in real life, digitally and in a form which the underlying network that Tangle utilizes, can understand.
The identity distribution process was developed to require as little human interaction possible in order to create and handle the invitation. The process, from the user’s point of view will now be described. For our example, let us say that two people, Tom and Mary, are currently using Tangle. Tom would like to invite Mary to a group he is currently a member of, but Tom’s Tangle is not yet aware of whom Mary is. Tom knows Mary’s email address so he informs Tangle that he would like to invite said email address to a specific group. Mary checks her email and sees that she has received an invitation to join a Tangle which Tom is a member of. Mary would like to join this Tangle so she double clicks on the file which was attached within the email and Tangle then handles the rest for Mary. She has interacted once with email, which is all she will have to do. Tom then receives an email saying that Mary has accepted his invitation, and he then double clicks on the file which was attached within that email. The process now completes and Mary is now a member of the Tangle Tom is a member of. Tom only interacted with his email once, as did Mary. All other interactions were performed by Tangle without user interaction.

Behind the scenes many things went on when describing the experience from Tom and Mary’s point of view. Tom’s first interaction to begin the process of inviting Mary to the Tangle required him to inform Tangle of Mary’s email address. Behind the scenes, Tangle used this information to send Mary an email with a file attached. The file attached is registered to be handled by Tangle and contains the following information: some data about the group Tom would like to invite Mary to such as the name, and Tom’s identity. When Mary handles this email and accepts the invitation, by opening the attached file, Tangle then creates a distribution group on Mary’s machine and uses Tom’s identity to create an
invitation to this distribution group for Tom. Tangle then emails information about this acceptance of the invitation, along with another attached file. Within this attached file is the invitation to Mary's distribution group for Tom. When Tom handles this email, by opening the attached file, he, unknown to him, joins Mary's distribution group and then queries Mary for her identity using that newly joined group. When Tangle receives Mary's identity it then creates an invitation to the original group which Tom would like Mary to join and then sends that invitation via the distribution group created. Mary's Tangle then receives this invitation and can finally join the group. Once this process is complete the distribution group is no longer needed and both parties can leave the group.

After this process has completed between Tom and Mary, either party can invite the other to another Tangle via the Tangle they are currently a member of and are not required at all to use email as the distribution mechanism.

An interesting part of Tangle which has yet to be introduced is the process required in order to remove someone from a Tangle. As it stands right now, no such process exists. This is due to the underlying network and how it structures users within groups. There is a distinction of two types of people within a peer group, an administrator and a user. The administrator has full rights to the group and can invite others to the group, where a user cannot invite others to the group. That is the main distinction between the two users. In order to allow all members of a group invitation rights, we had to make all users of Tangle administrators when they are invited to join a Tangle. Given this, in order to remove someone from the group one would expect that an administrator would have this power. There, however, is no functionality within the underlying networking to remove someone
from a group. This decision was made for many reasons, mainly because it introduces many new problems within a peer-to-peer network system such as WP2PN and it can be accomplished at the application level. For example, Tangle can manage more than those two types of users and when the highest level user (user with the most privileges) decides to remove another user then the application, Tangle, would be responsible for actually removing the user from the group. This is an easy process to perform from a developer's point of view, but it cannot be verified because the underlying networking does not allow for such verification since any application can use any peer group created by the WP2PN platform. Therefore a malicious user does not have to use Tangle in order to communicate with others who use Tangle within the peer groups. He/she would simply have to mimic the input and output of the system. Processes have been discussed thus far will discourage such interaction, mainly having Tangle encrypt the data sent over the peer group so that eavesdroppers would not be able to view it. This would stop separate applications from interacting with members of a Tangle in any way other than a replay attack (replaying messages already received). This, however, does not stop the user from being a member of the group within the underlying networking. Tangle can solve this problem by doing a verification step on members of the peer group to verify that they are in fact using Tangle before accepting records from them. This solves the issues of other applications being able to utilize the same peer groups which Tangle uses in the underlying framework.

As for the ability to remove members from a Tangle, one should start with allowing anyone to remove anyone else from the Tangle as anyone now has the ability to add anyone to the Tangle. The Tangles are created out of groups of trusted individuals and there is a
level of comfort and trust associated with such an assumption which can be exploited here. If someone is erroneously removed from a Tangle, he/she can just as easily be invited back into the Tangle as well.
CHAPTER 4: TANGLE EXTENSIONS

Tangle Extensions (TEs) provides unlimited extensibility for Tangle. TEs are skins for Tangle that define task optimized interfaces to particular data collections. TEs are separately developed programs that plug-in to Tangle to provide users with additional ways to manage and display information within a group of peers. TEs are the principal user interface that users of Tangle experience. This chapter focuses on the developer’s interface for new TEs and the process a developer goes through to create a TE.

The TE interface consists of an assembly shared between Tangle and the TE. This assembly contains common data types and structures which are used to communicate between the TE and Tangle. The basic approach to creating a TE is to add the shared assembly to the project space and implement a single interface, ITangleExtension. With this implementation, Tangle has the information necessary to add the TE to the list of usable extensions, allowing Tangle users to take advantage of the new extension. This process has been greatly simplified for this introduction and will be discussed in greater detail in the balance of the chapter.

Tangle has been created to work very closely with the .NET Framework, to allow developers to choose from the many languages which work within the .NET Framework itself. Once a .NET assembly has been created, any of the .NET Framework languages can be used to create a TE. Choices include VB.NET, C#, or Managed C++ at present; more languages may be supported by the .NET Framework in the future. If one is interested in working in other languages when developing a TE, a good starting place for information on such a subject would be the Mono Project [8]. The Mono Project is an open source
Implementation of the .NET Framework for other platforms other than Windows. It provides a good starting point for learning how to interact with the framework in non-standard ways.

Creating a Tangle Extension

Creating TEs must be as simple as possible, since new extensions are what will drive the development and use of Tangle. Creating a new TE is a simple, three step process:

1. implement the ITangleExtension interface in the shared assembly
2. fill in extension specific data such as a unique id, extension name, etc
3. implement a list of functionality for events received from Tangle and handle events sent to Tangle

Below we will walk through the creation of a basic TE, tChat, to provide an example of the TE creation process. The tChat extension is a simple extension which implements a group chat area. Using tChat peers can post messages to other peers who are currently online, in real-time. Eventually, more advanced features such as inline image posting, smiley faces, and even things such as voice or video data could easily be added to the extension, but for the sake of simplicity, we will outline only the barest functionality to allow us to focus on the definition process itself.

Step 1: Implementing ITangleExtension

The first task in creating a TE is to implement the ITangleExtension which is found inside of the assembly, Tangle.Extensions.dll. This assembly contains all of the shared data which can be used by both the extension and Tangle. In order to implement the interface, a developer simply creates a new class which uses the interface, ITangleExtension, as its base.
With this done, Tangle knows how to interact with the extension because Tangle understands how to interact with an ITangleExtension – regardless of the implementation details – so long as the required implementation exists. When testing a TE, one simply loads the dll into the Tangle system as a new Tangle Extension. This process has already been discussed in a previous chapter. The actual mechanics of adding the required assembly to the project when creating a TE will not be discussed here, as there are many different products and procedures which can be performed in order to accomplish this task.

In our tChat example, we create a class called tChatExtension which implements the ITangleExtension interface. The class tChatExtension is our main access point to communicate to Tangle and to receive information from Tangle. The tChat assembly includes an objectification of a chat message which includes items such as the identity of the message sender, the time the message was received, as well as the contents of the message itself. By doing this we will also be able to extend this message at a later date with different types with very little to no change in tChat itself. We then add a new type of chat message, based on the previously defined chat message, referred to as whisper message. A whisper message is a message sent to a single person, rather than the entire group of peers. The whisper message will be important later on when we show the difference between sending data directly to another peer and sending the data to the entire group in the form of a group record.

**Step 2: Filling in Required Data**

The next step in creating a TE is to fill in all of the required data. Tangle registers new TEs based on the unique id, of type System.Guid, associated with each extension. There
are many tools available which can create unique guid's, such as the *Create Guid* tool within Microsoft Visual Studio .NET [22]. There are other such unique ids which must be created. In order to add new data to the group the developer adds a new record to the group. Since records are strongly typed, each record type requires a unique type id. The developers of TEs are required to define types for each type of record which are to be added to the group.

There are two user interface (UI) tasks related to the required data which must be completed for the TE. The first is the menu which the extension will use, and the second is the main form which will be shown when a user of Tangle wishes to interact with the extension. The menu shows up via a right click-menu of a Tangle group and can be seen later in the tChat example. This menu provides a consistent access method for each Tangle extension. The contents of the menu are left up to the TE developer, but the menu structure is made up of a single menu item (type System.Windows.Forms.MenuItem) with as many sub-menu items as the developer of the extension requires.

The second UI task is the creation of a windows form for use by the extension. This is referred to as the *main form* for the TE. This form is used for direct interaction with the TE by a user of Tangle. The main form of the TE defines the user interface that the extension provides. There are two events defined within the ITangleExtension interface which deal with the main form. The event MainFormShow is received from Tangle and handled by the TE to display the form to the user. The event MainFormHide does just the opposite, hiding the main form from the user.

The last of the required data for the new TE consists of data useful to the user of Tangle. This data refers to things such as the name of the TE, a url where the user can find
more information on the TE, a short explanation of what the extension does, and an icon for the extension. There are two types of icons associated with a TE. The first icon, accessed via the *Icon* property within the ITangleExtension interface, is the main icon for the extension, and should be of size 64x64 pixels. The second icon, accessed via the *OverlayIcon* property, is a small icon, of size 16x16 pixels, which can be shown as an overlay to Tangle when there are pending tasks which can be performed by the user. This is useful when the main form of the extension is not shown and yet new information is available for the user to handle. The use of icons will be discussed further when defining the data used in our tChat example.

Consider the required data that must be filled in for our tChat example. First we define a unique id for tChat using the *Create Guid* tool in Visual Studio. We repeat the process to create unique id’s for the different record types we will be using in this TE, in this case the two record types: a chat message and a whisper message. We package the two newly created record handling id’s into a System.Collection.ArrayList and return that ArrayList of handled record types to Tangle via the *TypesHandled* property of the ITangleExtension interface. The TypesHandled property informs Tangle what types of records tChat is uniquely responsible for handling. All records of the types defined will be funneled to tChat.

Next we define tChat’s menu. Figure 11 shows the menu system in tChat. The menu consists of a System.Windows.Forms.MenuItem labeled ‘tChat’ and a sub-menu item labeled ‘Open tChat’. The purpose of the sub-menu item is self-explanatory, as it simply opens up the main form for tChat. This functionality does not happen for free however, and must be implemented within the event handler for the menu so that the main form is shown when the user clicks that menu item. Tangle accesses tChat’s menu via the *MenuItem* property defined
in the ITangleExtension interface therefore we must return the menu we just created within this property.

![Image](image.png)

**Figure 11:** The tChat menu accessible from the right click menu of the Tangle entitled, VRAC.

In Figure 11 above, someone clicking the 'Open tChat' sub-menu item expects that the main form for tChat will open. We must define the main form for tChat, so we will do that now. Figure 12 shows the main form for our tChat example. This form shows a chat window with all of the messages, who is currently online, and finally a space to write messages to be sent to the group of online members. An extension's main form should never be destroyed, only hidden. Clicking the 'x' in the top right corner of the form should not dispose of the form within .NET; it should simply hide it from view. This is accomplished by handling the OnClosing event for the form and setting the cancel argument to true and then calling the form's hide function.
The last pieces of required data for tChat are the icons Tangle will use to represent tChat. The Icon property is defined to return a specific icon of size 64x64 pixels and the OverlayIcon property is defined to return a specific icon of size 16x16 pixels. The overlay icon is used to show pending tasks to the user of Tangle. In our tChat example, we inform Tangle to show our overlay icon when new chat messages arrive and the main form for tChat is not currently visible. This overlay icon can be seen in Figure 13 as the yellow tChat icon displayed in the bottom right hand corner of the VRAC Tangle’s space. The actual features and reasoning for overlay icons have already been discussed. One needs to implement this feature in order for the TE to utilize the feature of overlay icons used for notification purposes.
Step 3: Receiving Events from Tangle and Talking to Tangle

The final step in creating a new TE is to implement the functions which handle events, calls made to the TE by Tangle. The first function which needs to be handled is the startup function which is called on a TE prior to its first use. The startup function takes two parameters, a string for the path where the TE can save data, and the identity of the user who is currently using Tangle. The startup function informs the extension that it is ok to load the data associated with the Tangle. There is a reciprocal function to startup called shutdown. The shutdown function informs the extension that the system is shutting down and any unsaved information should be saved to the path passed to the extension during startup.

The remaining event functions which must be implemented deal directly with tasks and events arising from the activity of the group of peers. The first of these functions deals with the current status of the Tangle. There are three different status types: listening, has connections, and disconnected. The Tangle is listening when the user is online and connected to the group, but no other peers are connected. The Tangle has connections when the user is connected to the group and there other peers also connected to the group. The
Tangle is disconnected when the user is not currently connected to the group. This function is useful when tasks need to be completed when an initial connection is made to the group, for example, updating old records with new records the Tangle may be holding.

The next event function is the TangleRecordChanged function. This function is invoked whenever a record on the Tangle has changed in some manner. The possible ways a record can change are: it was just created and is new, a record which is already on the Tangle has been updated in some way, a record has expired, or a record has been deleted. In all cases, a TangleRecord is passed along with the status of the record change to the function. The data property of the TangleRecord is only valid for the record being added and updated. When the record expires or is deleted, this property is not valid and instead the developer of the TE should use the id property associated with the TangleRecord in order to match to a record which could possibly be monitored. The record type and the id for the TangleRecord are always valid no matter what type of change has occurred.

The final Tangle to TE function which the developer must define is the TangleMemberStatusChanged function. This function informs the TE on the status of group members coming online and offline. The function sends in the identity of the Tangle member who has come online or offline and the status of that member. There are four different possible statuses which can be passed into this function: connected, disconnected, joined and left. Connected and disconnected correspond are generated when a member comes online or offline respectively. The other two statuses deal with members being added to or dropped from the group. The joined status means that the given identity has just joined the Tangle
through an invitation by another member of the group. The left status informs the group that the identity has been completely removed from the group and is no longer a member.

In our tChat example we use the startup function in order to load our chat history and the shutdown function to save the chat history to a file so it can be loaded the next time the tChat extension is started. The connection status function is not used in tChat as it does not matter to us if we are connected or not. If the user hits the send button and the user is not connected to the group, the message does not display in the chat history. Only messages received as records are displayed in the chat history. Since chat messages cannot be changed or removed after they have been sent, the only part of the receiving of a new record function we are concerned with is that when a new record has been added to the group. When this happens, tChat checks for the type of record it is, normal chat or whisper, and handles it accordingly. Finally, tChat is concerned with members coming online and offline so tChat implements the member status function. When a member comes online, he/she is added to the listing of online members and when he/she goes offline, the person is removed.

The next discussion focuses on how a TE talks to Tangle, or rather makes calls to the Tangle system in order to perform tasks and retrieve information from the system. This is accomplished through a relatively simple event system which is a part of the ITangleExtension interface. The event system allows for TEs to send data to Tangle and to retrieve needed information about the current state of the Tangle.

The first and most important event is also one of the most frequently used when creating a TE is the event which adds a new record to the Tangle. This event takes the type of record to create, the time for which this record is to expire, and the data to be held in the
record as parameters to the function. The type of record to create is of type System.Guid and directly corresponds to the types of records handled by the TE which has been talked about in the data portion of creating a TE. The time for which the record is to expire is of type System.DateTime. The data portion is an array of bytes which will be set as the data for the record to be created. This data can consist of anything the developer of the TE can dream up, but again, it should correspond to a type so that the data can be unpacked, so to speak, by others using this extension in the group of peers.

In addition to the Events raised by Tangle, there are a set of accessors that allow an extension to query the Tangle. These TE to Tangle functions retrieve data from Tangle at the request of the TE. The following are the accessors and the functionality they provide:

- RetrieveTangleName retrieves the name of the Tangle
- RetrieveMembers retrieves all the identities of the members of the Tangle and can be instructed to only retrieve the current online members
- IsOnline retrieves if the given identity is online or not
- RetrieveRecords retrieves all of the current records which can be handled by the TE and are currently on the Tangle

There is one additional function, RetrieveEndPoint, to retrieve an end point for a given online identity so that a separate, direct communication mechanism can be brokered by the TE to another peer. This mechanism is somewhat buggy in its current implementation. There are many factors which play into the success or failure of this function, especially with the relatively new introduction of Service Pack 2 (SP2) for the Windows XP platform dealing with issues within the built in firewall. At this point in time, it is recommended that a
separate direct connection be created through the group-wide messaging system at hand, for example, requesting the IP address of an identity through a type of record and then creating the connection based on this information. More work is required by this method, but this method also gives more power to the developer of the TE.

The final event to be defined, SendDataDirectly is used to send data directly from one peer to another. This is the main method and preferred way for sending small chunks of data directly to another peer, bypassing group-wide distribution of the data. SendDataDirectly takes the same parameters as the AddRecord event plus one more. The additional parameter is the identity of the peer to whom the data should be sent. SendDataDirectly is best suited for sending small amounts of data or if the developer does not care how and when the data is sent. If a large amount of data is to be sent or the developer needs control over the actual send/receive process, then the methods talked about previously for creating a manual direct connection to another peer should be used.

Undocumented Features

There are features of creating a TE which have been designed, but are not currently implemented within the Tangle system. These features have to do with indexing and search mechanisms within Tangle. While this incarnation of Tangle does not currently support indexing and search, that feature is the next step in its development. We will discuss the integration of search and indexing more in chapter 5, however, TE development has some provision for search even though Tangle, itself, does not currently use this functionality. Eventually, Tangle will expose an indexing mechanism for extensions in order to index data
being stored and used in the extension. Within the shared assembly, there is a control called SearchResult. This control contains information about a search result: a small icon or picture, the actual name of the result, and then a short description of the result.

Extension developers need only be concerned with implementing two functions to support search: Search, and OpenResult. The Search function takes a string query as a parameter and returns an ArrayList of SearchResult’s with the results of the query. This structure allows developer to display search results in any way he/she deems fit for the extension, or to simply return nothing if the developer does not want to deal with indexing. The OpenResult function informs the TE that the user would like to open a result received from the call to Search made on this extension. The OpenResult function takes a SearchResult as a parameter and the extension should handle this accordingly.
CHAPTER 5: FUTURE WORK

The storage of information on a personal computer which is to be accessed via some type of search engine is an emerging concept and is by no means close to the scale of information retrieval via the internet [10]. People who use the internet, however, are accustomed to using search engines in the process of information retrieval [10]. With the advent of desktop search applications, they can now include their personal information stores as part of the results of searches. The creation of desktop search engines is a natural outgrowth of the increase in personal data storage.

There are some shortcomings in using desktop search engines as a means of finding information. The basic set of problems in finding information via search engines on the internet are mirrored in that of desktop searching as well. For example, when someone searches for something, the usually have a goal in mind, or a specific domain or context defining what they are searching for [11]. Search engines do not account for such high level cognition and the user of the system must take time parsing the results further for useful information. It must be noted that the time spent parsing results is still better than it would be if no search mechanism were in place. The ultimate goal of a search engine (personal or otherwise) is to find information useful to the person performing the query. Checkland and Holwell said, in 1998, “The nature of an information system is to provide informational support to people as they carry out their intentional tasks.” [11, 12] In general, people perform tasks which are constrained by some context which provides a limiting factor on the useful information to that context [13].
Another significant shortcoming of desktop search engines is that there is no integrated way to share one’s acquired knowledge with others. The digital asset store that a user amasses is rooted within specific areas or domains that are personal to that user. These knowledge domains also typically involve interactions with groups of people, either at work, with friends, or with family members. The people a user interacts with will typically share interests and goals, and so have similar knowledge domains. Providing the ability to access knowledge within specific domains of interest among groups of peers is a significant enhancement to the Tangle concept. In fact, this search among peers is the principal feature of the first iteration of Tangle.

Clearly, the next stage of development for Tangle is the integration of a desktop search mechanism. The knowledge that one builds up automatically via a system such as Tangle will grow and become more valuable as time progresses and people build tight knit groups of peers who contain common goals within the groups created. Incorporating a context sensitive search into Tangle could leverage these pools of knowledge across the domains (Tangles) and link information between groups in unique and exciting ways.

A search and retrieval mechanism in Tangle would consist of three distinct parts: the query, the result, and finally the retrieval mechanism. The query mechanism is the more complicated of the two parts because of Tangle’s P2P architecture. There is no central server which can route a query to everyone within the group, and with Tangle, one can have multiple groups of peers with some people in more than one group. The problem reduces to designing an efficient mechanism to propagate a query across every peer whom one would be associated with via Tangle.
Apart from query distribution, there are many other considerations dealing with a query that will affect the perceived speed of the system. A mechanism proposed by Zeinalipour-Yazti, Kalogeraki, and Gunopulos, referred to as Intelligent Search Mechanism (ISM), provides a strategy for quickly and efficiently propagating a query through a P2P network [1]. The query propagation consists of four parts: a profiling structure used to profile each peer’s performance characteristics, a query similarity function to compare different search queries, a relevance ranking system to prioritize who gets sent the query first, and finally a search mechanism which actually sends the query to the peers. Using this type of system, Tangle could greatly reduce the amount of time associated with making a query to the system and receiving the results of that query.

Returning results after a query is the next step in the process. A cognitive model of just in time (JIT) data presentation proposed by Kester, Kirschner, Merrienboer, and Baumer [14] is useful in developing a JIT data presentation system for Tangle. The most important part of a JIT data presentation system would be deciding how the system should learn to predict and prioritize data for presentation to the users of Tangle. With this priority in place, one could help to reduce the costs associated with network latency and initial result presentation by sending higher priority information before lower priority information.

The unit responsible for prioritizing should learn to prioritize based on two types of skills, specifically recurrent skills, and non-recurrent skills [14]. In the case of Tangle, recurrent skills would be based on the idea that if a particular piece of information in one’s knowledge store is perceived as useful by many people, then this piece of information will be given a relatively higher priority in results of future queries.
Non-recurrent skills require more of a cognitive load and are based on ideas such as the context by which the information has entered the knowledge store. An example of non-recurrent skills within Tangle would be a higher priority being placed on a query from a peer given context A when the information was placed into the knowledge store via the same context, A. The identification of non-recurrent skills is much more complex and would be highly subjective based on the goals of individual users of Tangle. On the other hand, the process of implementing learning through recurrent skills would be relatively straightforward. Such an implementation would look at the number of times a particular piece of information within one's knowledge store was used by not only oneself but should also be affected by the number of times it shows up from query hits (originating both externally and internally) and the number of times those external hits lead to the data being used by that peer.

A problem with P2P networks is efficiently retrieving information shared by other peers within that network [1, 16]. Problems can also arise when large files, most notably that of media (video, music, etc) are desired to be transferred from one peer to another peer [16]. Most P2P systems today require the client to first download the entire file before using it [17]. While this is acceptable for documents such as Word files, emails, or pdf's, it can be quite undesirable when attempting to obtain large media files, such as video, where streaming would be possible within a client-server paradigm. The concern here does not lie with the documents which must be fully downloaded to be used, which usually are relatively small (perhaps a few megabytes), especially when compared to that of stream-able media (can approach sizes on the order of gigabytes).
A real opportunity for improvement in Tangle is to incorporate a form of peer to peer streaming for large files. Protocols already exist for distributed content delivery such as BitTorrent [15]. None of these protocols, however, allow for the streaming of media, or playing media before the entire download has been completed. The architecture discussed by Hefeeda, Bhargava, and Yau for on-demand media streaming via P2P networks could be used by the system [17]. This hybrid architecture takes into account many items in order to accomplish the streaming of media from aggregated peers and based on their results, the design works well. The basic outline of the design is to prioritize the download of information from the beginning of the file and stream this information as it is needed from different sources. This not only has the possibility of drastically improving performance but also distributes the workload in a more manageable manner for the overall system.

Another part of Tangle which is being left to future work is that of group size considerations on the quality and quantity of information present to a user of Tangle. Some research has been done, mainly dealing with the areas of group support systems and collaborative environments [18] in order to formulate a quantitative measure of group sizes and the amount of useful information which can be obtained using a system similar to Tangle. These questions are waiting to be answered by a prototype of Tangle along with testing the design goals set forth by this research.
APPENDIX: WINP2P

Namespace List

The namespaces specified in this document are:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinP2P</td>
<td>WinP2P</td>
</tr>
<tr>
<td>WinP2P.Grouping</td>
<td>WinP2P</td>
</tr>
<tr>
<td>WinP2P.Identity</td>
<td>WinP2P</td>
</tr>
</tbody>
</table>

Namespace: WinP2P.Grouping

WinP2P.Grouping Type List

Enumerations

<table>
<thead>
<tr>
<th>Type</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFailedReason</td>
<td>An attempt at a connection with a peer group has failed for some reason.</td>
</tr>
<tr>
<td>DirectConnectionStatus</td>
<td>The status of a direct connection</td>
</tr>
<tr>
<td>GroupRole</td>
<td>The different types of roles a member of a group can have.</td>
</tr>
<tr>
<td>MemberChangeType</td>
<td>A member of the peer group has changed in some manner.</td>
</tr>
<tr>
<td>PeerGroupFlags</td>
<td>These are used to specify various peer group membership settings.</td>
</tr>
<tr>
<td>PeerGroupStatus</td>
<td>The status of a peer group has changed.</td>
</tr>
<tr>
<td>PeerRecordFlag</td>
<td>Flags that are associated with a peer record.</td>
</tr>
<tr>
<td>RecordChangeType</td>
<td>A record on the peer group has changed in some way.</td>
</tr>
</tbody>
</table>

Delegates

<table>
<thead>
<tr>
<th>Type</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFailedDelegate</td>
<td>There was an error when attempting to connect to a group.</td>
</tr>
<tr>
<td>DirectConnectionStatusDelegate</td>
<td>A direct connection has changed.</td>
</tr>
<tr>
<td>GroupPropertiesChangeDelegate</td>
<td>The properties of a group has changed.</td>
</tr>
<tr>
<td>GroupStatusChangeDelegate</td>
<td>The status of a group has changed.</td>
</tr>
<tr>
<td>IncomingDirectConnectionDataDelegate</td>
<td>There is data incoming from a direct connection.</td>
</tr>
<tr>
<td>MemberChangeDelegate</td>
<td>The status of a member of a peer group has changed.</td>
</tr>
<tr>
<td>RecordChangeDelegate</td>
<td>A record within the group has changed.</td>
</tr>
</tbody>
</table>
## Classes

<table>
<thead>
<tr>
<th>Type</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectConnection</td>
<td>This class encapsulates a direct connection that is made with another peer in a group.</td>
</tr>
<tr>
<td>GroupingException</td>
<td>This class represents an exception which is thrown when working within the WinP2P.Grouping namespace</td>
</tr>
<tr>
<td>PeerGroup</td>
<td>This class represents a peer group within the peer-to-peer API.</td>
</tr>
<tr>
<td>PeerRecord</td>
<td>This class encapsulates a record held on a peer group</td>
</tr>
</tbody>
</table>

## WinP2P.Grouping Enumerations

### PeerGroupFlags Enumeration

#### Summary

public enumeration PeerGroupFlags

These are used to specify various peer group membership settings.

This type has the following attributes: System.SerializableAttribute

#### Enumeration Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeferExpiration</td>
<td>Group records are not expired until the peer connects to the group.</td>
</tr>
<tr>
<td>DisablePresence</td>
<td>The peer presence system is prevented from automatically publishing presence information</td>
</tr>
<tr>
<td>MemberDataOptional</td>
<td>The peer's member data is only published when the peer performs an action, such as adding a new record.</td>
</tr>
<tr>
<td>None</td>
<td>No flags specified.</td>
</tr>
</tbody>
</table>

### MemberChangeType Enumeration

#### Summary

public enumeration MemberChangeType

A member of the peer group has changed in some manner.

#### Enumeration Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>A member has connected to the group.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>A member has disconnected from a group.</td>
</tr>
<tr>
<td>Joined</td>
<td>A peer has just joined the group.</td>
</tr>
</tbody>
</table>
RecordChangeType Enumeration

Summary
public enumeration RecordChangeType
A record on the peer group has changed in some way.

Enumeration Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added</td>
<td>A new record has just been added to the peer group.</td>
</tr>
<tr>
<td>Deleted</td>
<td>A record on the peer group has been deleted and is no longer valid.</td>
</tr>
<tr>
<td>Expired</td>
<td>A record on the peer group has expired.</td>
</tr>
<tr>
<td>Updated</td>
<td>A record on the peer group has been updated in some way.</td>
</tr>
</tbody>
</table>

PeerGroupStatus Enumeration

Summary
public enumeration PeerGroupStatus
The status of a peer group has changed.

Remarks
PeerGroupStatus.Disconnected is NOT part of the underlying API. It was put it so that an event is fired when you disconnect from a peer group.

Enumeration Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnected</td>
<td>This will occur upon a successful disconnection from a peer group.</td>
</tr>
<tr>
<td>HasConnections</td>
<td>There is more than one peer (including yourself) connected to the group.</td>
</tr>
<tr>
<td>Listening</td>
<td>In this state, you (you’re identity) is the only person connected to the group and are waiting for connections</td>
</tr>
</tbody>
</table>

ConnectionFailedReason Enumeration

Summary
public enumeration ConnectionFailedReason
An attempt at a connection with a peer group has failed for some reason.

**Enumeration Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoMemberConnections</td>
<td>There were no member connections available.</td>
</tr>
<tr>
<td>NoMembersFound</td>
<td>There were no available peers in the peer group to connect to.</td>
</tr>
<tr>
<td>UnableToListen</td>
<td>The peer was unable to receive connection data for an unspecified reason.</td>
</tr>
</tbody>
</table>

**PeerRecordFlag Enumeration**

**Summary**

`public enumeration PeerRecordFlag`

Flags that are associated with a peer record.

**Enumeration Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autorefresh</td>
<td>The record is automatically refreshed when it is about to expire.</td>
</tr>
<tr>
<td>Deleted</td>
<td>The record is marked to be deleted.</td>
</tr>
</tbody>
</table>

**GroupRole Enumeration**

**Summary**

`public enumeration GroupRole`

The different types of roles a member of a group can have.

**Enumeration Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>This role can issue invitations, issue credentials, and renew the GMC of other administrators, as well as behave as a member of the peer group.</td>
</tr>
<tr>
<td>Member</td>
<td>This role can publish records to the group database.</td>
</tr>
</tbody>
</table>

**DirectConnectionStatus Enumeration**

**Summary**

`public enumeration DirectConnectionStatus`

The status of a direct connection
**Enumeration Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>A new direct connection has been made.</td>
</tr>
<tr>
<td>ConnectionFailed</td>
<td>The direct connection has been closed.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>The attempted direct connection failed.</td>
</tr>
</tbody>
</table>

**WinP2P.Grouping Delegates**

**MemberChangeDelegate Delegate**

**Summary**

```csharp
class MemberChangeDelegate
{
    public delegate MemberChangeDelegate
}
```

The status of a member of a peer group has changed.

**GroupStatusChangeDelegate Delegate**

**Summary**

```csharp
class GroupStatusChangeDelegate
{
    public delegate GroupStatusChangeDelegate
}
```

The status of a group has changed.

**GroupPropertiesChangeDelegate Delegate**

**Summary**

```csharp
class GroupPropertiesChangeDelegate
{
    public delegate GroupPropertiesChangeDelegate
}
```

The properties of a group has changed.

**RecordChangeDelegate Delegate**

**Summary**

```csharp
class RecordChangeDelegate
{
    public delegate RecordChangeDelegate
}
```

A record within the group has changed.
ConnectionFailedDelegate Delegate

Summary
public delegate ConnectionFailedDelegate
There was an error when attempting to connect to a group.

DirectConnectionStatusDelegate Delegate

Summary
public delegate DirectConnectionStatusDelegate
A direct connection has changed.

IncomingDirectConnectionDataDelegate Delegate

Summary
public delegate IncomingDirectConnectionDataDelegate
There is data incoming from a direct connection

WinP2P.Grouping Classes

PeerRecord Class

Summary
public class PeerRecord
This class encapsulates a record held on a peer group
This type has the following attributes: System.SerializableAttribute

Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerRecord()</td>
<td>public</td>
<td>Nothing special.</td>
</tr>
</tbody>
</table>

Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes : String</td>
<td>public</td>
<td>Gets or Sets the XML representation of the attributes associated with this peer group.</td>
</tr>
</tbody>
</table>
### Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
</table>

### PeerGroup Class

**Summary**

public class PeerGroup

This class represents a peer group within the peer-to-peer API.

### Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerGroup()</td>
<td>public</td>
<td>The constructor for a PeerGroup.</td>
</tr>
</tbody>
</table>

### Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier : String</td>
<td>public</td>
<td>Gets or Sets the classifier of this PeerGroup.</td>
</tr>
<tr>
<td>Cloud : String</td>
<td>public</td>
<td>Gets or Sets the cloud of this PeerGroup.</td>
</tr>
<tr>
<td>Comment : String</td>
<td>public</td>
<td>Gets or Sets the comment of this PeerGroup.</td>
</tr>
<tr>
<td>Connected : Boolean</td>
<td>public</td>
<td>Gets the status of us being connected to the peer group or not</td>
</tr>
<tr>
<td>Creator : Identity</td>
<td>public</td>
<td>Gets or Sets the creator of this PeerGroup.</td>
</tr>
<tr>
<td>Flags : PeerGroupFlags</td>
<td>public</td>
<td>Gets or Sets the flags of the PeerGroup.</td>
</tr>
<tr>
<td>FriendlyName : String</td>
<td>public</td>
<td>Gets or Sets the friendly name of this PeerGroup.</td>
</tr>
<tr>
<td>MemberDataLifetime</td>
<td>public</td>
<td>Gets or Sets the member data lifetime of this PeerGroup.</td>
</tr>
</tbody>
</table>
Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRecord() : Void</td>
<td>public</td>
<td>Add a new record to the peer group.</td>
</tr>
<tr>
<td>AddRecord() : Void</td>
<td>public</td>
<td>Add a new record to the peer group.</td>
</tr>
<tr>
<td>AddRecord() : Void</td>
<td>public</td>
<td>Add a new record to the peer group.</td>
</tr>
<tr>
<td>Connect() : Void</td>
<td>public</td>
<td>This function will initiate a connection with a peer group. After this function returns successfully, events are fired from the peer group.</td>
</tr>
<tr>
<td>CreateInvitation() : String</td>
<td>public</td>
<td>Creates an invitation for someone to join this peer group.</td>
</tr>
<tr>
<td>DeleteRecord() : Void</td>
<td>public</td>
<td>Deletes a record from the peer group.</td>
</tr>
<tr>
<td>Disconnect() : Void</td>
<td>public</td>
<td>This function disconnects from a peer group. You will no longer receive any data from the group, or send any data to the group.</td>
</tr>
<tr>
<td>OpenDirectConnection() : Void</td>
<td>public</td>
<td>This function will open up a direct connection between you and the peer specified in identity.</td>
</tr>
<tr>
<td>ResolvePeer() : SocketAddress</td>
<td>public</td>
<td>Resolves the IPv6 address for a given identity within the PeerGroup.</td>
</tr>
<tr>
<td>RetrieveGroupMembers() : Hashtable</td>
<td>public</td>
<td>Retrieves the members of the peer group.</td>
</tr>
<tr>
<td>RetrieveRecords() : Hashtable</td>
<td>public</td>
<td>Retrieves a Hashtable containing all of the PeerRecord's of type, recordType, within the PeerGroup.</td>
</tr>
<tr>
<td>SetGroupProperties() : Void</td>
<td>public</td>
<td>This function will update/change the group's properties within the P2P system. This function should be called after changing the internal Properties of this PeerGroup.</td>
</tr>
<tr>
<td>ToString() : String</td>
<td>public</td>
<td>Returns the Friendly Name of this PeerGroup.</td>
</tr>
</tbody>
</table>

GroupingException Class

Summary

public class GroupingException : WinP2PException, ISerializable

This class represents an exception which is thrown when working within the WinP2P.Grouping namespace

Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupingException()</td>
<td>public</td>
<td>Basic constructor for a GroupingException</td>
</tr>
</tbody>
</table>
Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DirectConnection Class

Summary

public class DirectConnection

This class encapsulates a direct connection that is made with another peer in a group.

Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectConnection()</td>
<td>public</td>
<td>Basic constructor for this class.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectedTo</td>
<td>public</td>
<td>Gets the Identity that the direct connections is with.</td>
</tr>
<tr>
<td>Identity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConnectionID : Int64</td>
<td>public</td>
<td>Gets the connection id for this DirectConnection. This can be used for comparative purposes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseConnection() : Void</td>
<td>public</td>
<td>Closes the connection with the peer. This invalidates all data within this class and this class can no longer be used for any direct connection items.</td>
</tr>
<tr>
<td>sendData() : Void</td>
<td>public</td>
<td>Sends data to the peer connected within this DirectConnection.</td>
</tr>
</tbody>
</table>

Namespace : WinP2P

WinP2P Type List

Classes

<table>
<thead>
<tr>
<th>Type</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2PManager</td>
<td>This function begins interaction with the Peer-to-Peer service running in Windows. This class is a singleton and can therefore not be instantiated normally. Call</td>
</tr>
</tbody>
</table>
WinP2P Classes

P2PManager Class

Summary

public class P2PManager

This function begins interaction with the Peer-to-Peer service running in Windows. This class is a singleton and can therefore not be instantiated normally. Call P2PManager.Instance() to create an instance of this class.

Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance: P2PManager</td>
<td>public</td>
<td>This function creates and returns the single instance of a P2PManager</td>
</tr>
</tbody>
</table>

Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateNewIdentity(): Identity</td>
<td>public</td>
<td>Creates a new identity on this machine.</td>
</tr>
<tr>
<td>DeleteIdentity(): Void</td>
<td>public</td>
<td>This function removes an identity from the system.</td>
</tr>
<tr>
<td>ExportIdentity(): String</td>
<td>public</td>
<td>This function is used to create an encrypted XML representation of this identity for use in moving the identity to another computer.</td>
</tr>
<tr>
<td>ImportIdentity(): Identity</td>
<td>public</td>
<td>This function is used to import an identity which has been exported using WinP2P.Identity.Identity.Export(...).</td>
</tr>
<tr>
<td>RetrieveIdentities(): ArrayList</td>
<td>public</td>
<td>Retrieves the identities that are currently on the system.</td>
</tr>
<tr>
<td>RetrieveIdentitiesAsHash(): Hashtable</td>
<td>public</td>
<td>Retrieves the identities that are currently on the system.</td>
</tr>
<tr>
<td>Shutdown(): Void</td>
<td>public</td>
<td>This function shuts down the Peer-to-Peer service and de-allocates any resources it was using.</td>
</tr>
<tr>
<td>Startup(): Void</td>
<td>public</td>
<td>This function starts the service. This must be called before calling ANY other Peer-to-Peer functions.</td>
</tr>
</tbody>
</table>

WinP2PException Class

Summary

public class WinP2PException : Exception, ISerializable

The base class for exceptions thrown from within the WinP2P namespace.
Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinP2PException()</td>
<td>public</td>
<td>Basic constructor for a WinP2PException</td>
</tr>
<tr>
<td>WinP2PException()</td>
<td>public</td>
<td>Exception that takes a string message.</td>
</tr>
<tr>
<td>WinP2PException()</td>
<td>public</td>
<td>An Exception which has a message and contains an inner exception.</td>
</tr>
</tbody>
</table>

Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
</table>

Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
</table>

Namespace: WinP2P.Identity

WinP2P.Identity Type List

Classes

<table>
<thead>
<tr>
<th>Type</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>This class represents the identity of a user.</td>
</tr>
<tr>
<td>IdentityException</td>
<td>This class represents an exception which is thrown when working within the WinP2P.Identity namespace</td>
</tr>
</tbody>
</table>

WinP2P.Identity Classes

Identity Class

Summary

public class Identity : ISerializable

This class represents the identity of a user.

This type has the following attributes: System.SerializableAttribute

Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity()</td>
<td>public</td>
<td>Constructor for a WinP2P.Identity.Identity.</td>
</tr>
<tr>
<td>Identity()</td>
<td>public</td>
<td></td>
</tr>
</tbody>
</table>

Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FriendlyName</td>
<td>public</td>
<td>Gets or Sets the friendly name of this Identity. The 'set' function will NOT update</td>
</tr>
<tr>
<td>String</td>
<td>the identity's friendly name in the peer-to-peer API and is only here for creating a new Identity.</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>PeerName : String</td>
<td>Gets or Sets the peer name of this Identity. The 'set' function will NOT update the identity's peer name in the peer-to-peer API. This function is RARELY ever used from outside of this class.</td>
<td></td>
</tr>
</tbody>
</table>

### Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString() : String</td>
<td>public</td>
<td>This function returns the XML representation of this identity. This is used to send to other people for invitations and other Peer-to-Peer actions requiring an XML representation of an identity.</td>
</tr>
<tr>
<td>CreateGroup() : PeerGroup</td>
<td>public</td>
<td>This function creates a new group within this identity with default settings.</td>
</tr>
<tr>
<td>CreateGroup() : PeerGroup</td>
<td>public</td>
<td>This function creates a new group within this identity with default settings.</td>
</tr>
<tr>
<td>DeleteGroup() : Void</td>
<td>public</td>
<td>This function deletes a group from the peer-to-peer infrastructure.</td>
</tr>
<tr>
<td>GetObjectData() : Void</td>
<td>public</td>
<td></td>
</tr>
<tr>
<td>JoinGroup() : PeerGroup</td>
<td>public</td>
<td>This function is used to join a group in which you have received an invitation.</td>
</tr>
<tr>
<td>RetrievePeerGroups() : ArrayList</td>
<td>public</td>
<td>This function retrieves the groups which are associated with this identity.</td>
</tr>
<tr>
<td>RetrievePeerGroupsAsHash() : Hashtable</td>
<td>public</td>
<td>This function retrieves the groups which are associated with this identity.</td>
</tr>
<tr>
<td>ToString() : String</td>
<td>public</td>
<td>This function returns the friendly name of this Identity.</td>
</tr>
</tbody>
</table>

### IdentityException Class

#### Summary

public class IdentityException : WinP2PException, ISerializable

This class represents an exception which is thrown when working within the WinP2P.Identity namespace

### Constructor Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdentityException()</td>
<td>public</td>
<td>Basic constructor for an IdentityException</td>
</tr>
<tr>
<td>IdentityException()</td>
<td>public</td>
<td>Exception that takes a string message.</td>
</tr>
<tr>
<td>IdentityException()</td>
<td>public</td>
<td>An Exception which has a message and contains an inner exception.</td>
</tr>
</tbody>
</table>

### Property Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
</table>
Method Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Summary</th>
</tr>
</thead>
</table>


REFERENCES CITED

[1] D. Zeinalipour-Yatzi, V. Kalogeraki, D. Gunopulos, Exploiting locality for scalable information retrieval in peer-to-peer networks, Information Systems -Article In Press-
[18] B. Mennecke, Using group support systems to discover hidden profiles: an examination of the influence of group size and meeting structures on information sharing and decision quality, Int. J. Human-Computer Studies 47 (1997) 387-405

[19] Application/Protocol, Gnutella

[20] Application, Kazaa
http://www.kazaa.com/ (date accessed: 5 May 2005)


BIBLIOGRAPHY

E. Gamma, R. Helm, R. Johnson, J. Vlissides, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston.

