



# nTEGRATOR

## nTegrator: An Introduction

A White Paper that introduces and explains the nTegrator product.

© nBoundary Software Inc.

No reproduction without permission.

*[For a description of nTegrator's competitive position in the marketplace see the companion white paper: "nTegrator: Market Position".]*

*This page intentionally left blank*

## Contents

Background .....	1
Product Overview .....	3
Objects .....	5
Data Sources.....	8
Collections.....	9
Messages .....	10
Distribution.....	11
Security .....	12
Footprint & Cache.....	13
nTegrator Explorer .....	14
Application Development .....	15
NewTech Cleaning Services (nTegrator In Action).....	16
Comparative Efficiency.....	23
Product Status .....	24
Programmer's Note .....	25

*Note: This document is intended for an audience that is generally knowledgeable about computing and application development, but it is not directed at programmers. In some places it uses analogy and simplification as part of the explanation. Actual implementation of some features may differ from how they are explained in this document, as described in the section 'Programmer's Note'.*

*This page intentionally left blank.*

## **Background**

In general, three types of tool are available for developing information systems:

- end-user tools such as office products (word processors, spreadsheets, etc.) and packaged software such as accounting and personnel systems;
- custom applications developed by programmers;
- total packaged solutions -- integrated suites of applications targeted at an enterprise's entire information management needs.

End-user tools and other packaged software can be inexpensive and often lend themselves to quite sophisticated applications. But training requirements can be heavy, product adaptation and system development can take users away from their regular work for extended periods, and the tools usually lack access to external corporate data.

Custom applications may accurately target specific business needs but are expensive to develop and maintain. They require the infrastructure of an IT department or the cost of external consultants.

Total solutions are extremely expensive both to buy and to implement and have a mixed record of success at accomplishing their aims.

In recent years, much effort has been devoted to 'enterprise application integration' -- making corporate information managed by one system available to end-user tools and other applications. Most such solutions involve the use of some kind of translator, a product that reads data from multiple sources and makes it available to other software in a standard format. These EAI 'hubs' are good as far as they go but they have a number of disadvantages:

- they tend to be expensive;
- they usually require a dedicated computer or at least a significant portion of one -- a system that uses the hub's capabilities has to physically access that computer;
- they rarely offer access to local data sources such as personal data stores and contact lists;
- they lack flexibility to easily add new data sources;
- they typically require a fair amount of installation and support effort;
- they often have to be customized to every individual application that wishes to access them.



## *An Introduction*

nTegrator offers a completely different approach to the same problem.

nTegrator is a “programmer-assisted end-user” tool. Developing an nTegrator application usually requires programmer involvement – albeit far less than a traditional custom-developed solution – but can otherwise be done by any technically proficient end user.

nTegrator applications are built out of components. Each component encapsulates one small part of the application’s intelligence. Components communicate with each other by passing messages. No intermediate or ‘translator’ product is needed.

nTegrator itself consists of two parts:

- nTegrator Explorer, the tool used for application development;
- the nTegrator engine, a memory-resident program used for application operation.

Applications built using nTegrator are developed very rapidly, typically requiring from 50% to 95% less code than a comparable custom application, but with all the benefits and few of the drawbacks of such a solution.

Although they operate on Windows devices, nTegrator applications are ‘agnostic’ as far as the source and format of data are concerned.

This document explains nTegrator and how it works. nTegrator’s application is limited only by the reader’s imagination.



## **Product Overview**

nTegrator is a toolkit and engine for rapidly and simply developing and enabling business information systems that run under the Windows operating system. nTegrator applications can easily be built to provide users with exactly and only the information they need, in exactly the form in which it is needed.

nTegrator may be used for single-user personal applications that draw data from existing data stores or for medium-scale production systems.

Every 'point of intelligence' in an nTegrator application is an object. nTegrator objects are programs, each made up of two components. The components themselves are also programs.

Components may be developed using anything from simple scripts such as XSL templates to VBScript to C++. Technically proficient users can produce many scripted components. Programmers are usually required for complex components that necessitate development using true programming languages.

A number of standard components are provided with nTegrator to perform common tasks such as reading from ODBC databases.

An nTegrator application may be distributed across multiple Windows devices provided only that the devices are connected through TCP/IP networks. The application can be quite arbitrarily divided among multiple computers with no change to the underlying architecture. This 'geographically irrelevant' architecture allows great flexibility in implementing applications in such a way as to optimize the features of each computer used.

Data in an nTegrator application may be drawn from any database, in any format, anywhere. Application results may be displayed in or through end-user products such as Excel or Internet Explorer.

Every nTegrator object is permission protected.

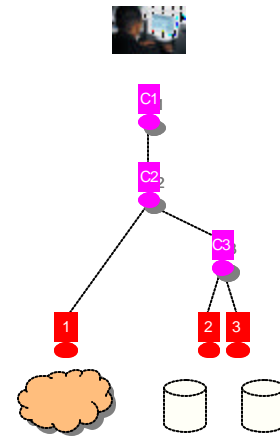
nTegrator leverages the standardization implied by protocols such as SOAP (simple object access protocol) and XML (extensible mark-up language). All data in an nTegrator application is passed from object to object in the form of XML-format messages. Any given XML message in an nTegrator application may contain data, commands, or any mix of both. Messages passed between objects that reside on different computers are automatically encrypted.

Any object or its component parts may be reused. If a complete object is reused, new functionality may be added to it without recompiling the object. Any new functionality so added is 'visible' only to the new application; the object's original functions remain unchanged in the application of which it was originally part.

nTegrator applications are built in four steps:

1. The business process is defined, perhaps using a modeling tool.
2. The model's components are created as 'collections' using nTegrator Explorer.
3. Object components are either developed or selected from stock.
4. Components are then dragged and dropped into objects using nTegrator Explorer.

nTegrator applications comprise hierarchical collections of objects. An application's highest-level object initiates and controls the application's operation. Lowest level objects generally perform input/output functions. Objects in between control logic flow within the application. Any object at any level can also perform one or more aspects of the application's underlying business rules. In general, business rules are processed at the lowest level possible. This permits organizing application systems by a top-down process of 'delegation'.



No networking knowledge is required on the part of a system's developer.

Needing as little as 200K, the nTegrator engine runs unobtrusively in any Windows device. Only one copy of the engine is required in each computer; it manages as many objects as required, no matter how many applications those objects are a part of.

nTegrator objects are stored in databases and accessed through nTegrator's own in-memory cache. A limited number of objects may be held in cache alone, so that nTegrator can operate on devices that have no external storage.

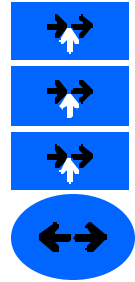
nTegrator's small footprint limits the likelihood of the process being paged in and out of memory by the operating system. Since most paging activities involve disk access, this helps to optimize processing times.

## Objects

nTegrator objects are of two types: Interface Objects and Collection Objects.

Interface Objects read from or write to data stores that are external to the nTegrator application.

- If the object's role is to read data, its Data Transport component reads the data and immediately converts it to an XML message.
- If the object's role is to write data, the Data Transport receives the data as part of an XML message and converts it to the external format before writing it.




An nTegrator object with one Data Transport and three Agents

Collection Objects act as switches for XML messages. The object's Data Transport routes the message in some way.

Any attached Agents act upon either the message (for example, determining which object to route it to next) or its contents (for example, performing some calculation on the data contained within the message).

### Data Transport

 A Data Transport is a program that handles data but does not modify it (except for converting it to or from XML).

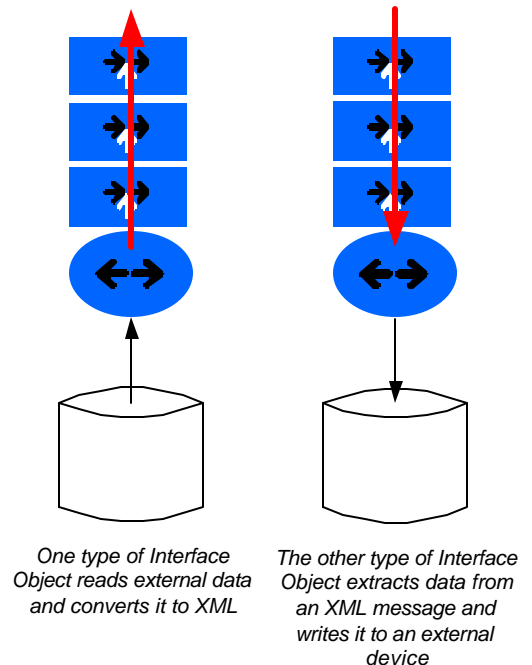
A Data Transport that is part of an Interface Object either reads data that is external to the application or writes data to an external device.

If the Data Transport is designed to read data, it converts that data to XML before introducing it to the nTegrator environment.

If the Data transport is designed to write data, it extracts that data from an XML message passed to it by another object.

A Data Transport that is part of a Collection Object acts as a switch.

Most Collection Objects (except for the highest-level object



in a system) have one superior object and one or more subordinate objects.

If the Data Transport in a Collection Object receives an XML message from its superior, it routes it to whichever subordinate is appropriate. This may be based either on the path selected by the highest-level object in the application or upon the content of the message.

If the Data Transport in a Collection Object receives an XML message from one of its subordinates, it routes it to its superior.

A Data Transport is a compiled program or a script.

## Agent



An Agent is a program that acts upon the data in an XML message.

Agents are 'attached' to Data Transports to form Objects. Every object has one and only one Data Transport but may have zero, one, or many Agents. This applies to both types of object: Collection Objects and Interface Objects.

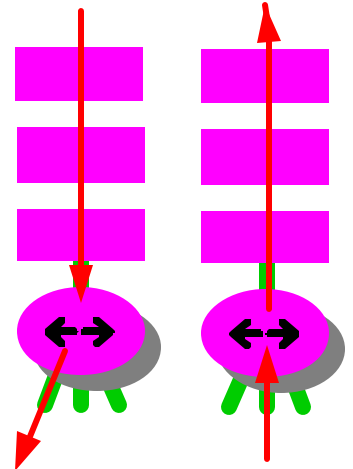
An Agent may contain application logic, business logic, or any combination of both. One Agent may be as simple or as complex as the application and its designer determine.

As a rule of thumb :

- Business logic is best handled at the lowest possible level in the collection hierarchy that makes up the application.
- Each Agent should perform one and only one business task. The simpler the Agent, the easier it is to develop, test, maintain, and reuse. Complex logic can usually be implemented using multiple Agents, all of which can, if necessary, be made part of one object.
- One Agent should contain system logic or business logic but not both.

An Agent may perform any task that any other program would do:

- ... analyze data
- ... change data
- ... make decisions
- ... route data
- ... combine data



*Collection Objects act as a switch, directing XML messages upwards and downwards through the hierarchy.*

- ... perform calculations
- ... format data
- ... etc.

An Agent recognizes when it is supposed to act on a message and when it is supposed to ignore that message.

An Agent is a compiled program or a script.

## **Non-Compilation**

When Data Transports and Agents are formed into objects, no compilation of the object is required.

Similarly, when new Agents are added to existing objects, no compilation of the object is required.

This unique feature is the primary reason why nTegrator applications are so easy to build, test, maintain, and modify. It is the subject of a pending US patent.

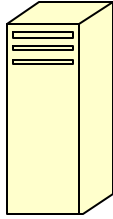
## **Reuse**



Any Data Transport, Agent, or complete object may be reused, either in the same application or by another application.

Additional Agents may be dropped onto reused objects. These Agents are visible only to the new application, not to the application that originally used the object.

## **Data Sources**



The Data Transport component of an Interface Object accesses data that is external to an nTegrator application. One such Data Transport will access one – and only one – external data source.

nTegrator is supplied with a standard Data Transport that accesses ODBC databases such as SQL Server and Oracle and that converts the data to XML format. The user has only to provide the appropriate parameters that identify the data required.

Similarly, a standard Data Transport supplied with nTegrator simplifies access to COM-aware databases and converts the data to XML.

Custom Data Transports have to be developed by the user to access other data sources and convert the data to XML. Typically, these would be written as VBScripts or as C++ programs. It is normal that the more of these Data Transports that are written, the more of the code that becomes boilerplate, able to be copied into new Data Transports.

Data Transports need not necessarily always access data stored on Windows devices. A Data Transport may utilize a communications protocol that supports the exchange of data with a non-Windows device. It will still do the necessary XML conversion.

Over time, more standard Data Transports will be developed, thus minimizing the user's need to develop custom Data Transports.

## Collections

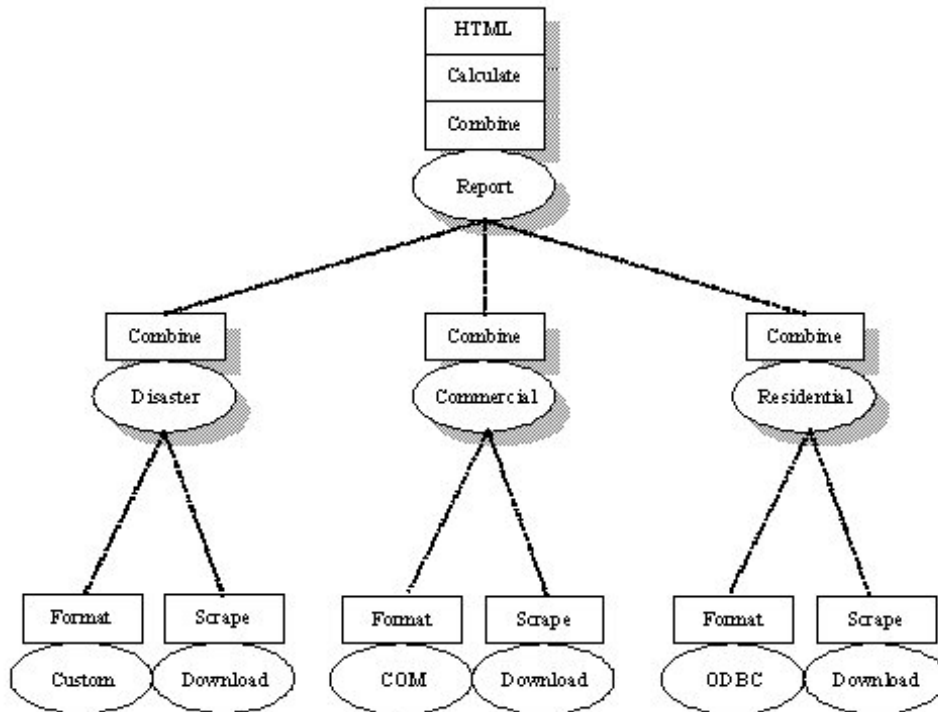
Every point of intelligence in an nTegrator application is an object. Objects are ordered into Collections. Collections determine the structure of the application.

Every application consists of a hierarchical “collection of collections”. The highest level collection, sometimes referred to as “C-zero”, has ultimate control over the entire application.

When a user double clicks the icon for an nTegrator application, he “enters” collection C-zero. C-zero then initiates the process of creating the transactions that make up the application.

Lower-level collections (C1, C2, C3...) repeatedly pass control to still lower-level collections (C11, C12, C13; C21, C22; C31, C32, C33...), until control reaches an Interface Object. This object typically reads data, converts it to XML, and starts the process of returning control back up the hierarchy.

Along the way, Agents attached to the Interface Object and the various Collection Objects act on the data to perform the business logic of the application.



*This is a typical application hierarchy. Collection C-zero is called “Report” (the same name as its Data Transport). Other collections are “Disaster”, “Commercial”, and “Residential”. There are six low-level Interface Objects. The actual system is described later in the White Paper.*

## Messages

All commands and data within an nTegrator application system take the form of XML messages.

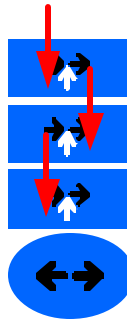
In general – but with exceptions – commands and parameters are sent ‘down’ the hierarchy and data is sent ‘up’.

Any given XML message can contain commands, data, or both.

A message traveling down through the hierarchy to an object is exposed to attached Agents before the object’s Data Transport routes it. A message traveling in the other direction is handled by the Data Transport before it is exposed to any attached Agents.

Attached Agents act upon the message in the order in which the Agents are attached.

```
<?xml version="1.0" ?>
<xmltest>
  <Source1>
    <record loc = 14>
      <Author>Adams, Pat</Author>
      <Phone>(403)555-1234</Phone>
    </record>
  </Source1>
  <Source2>
    <record key = 77>
      <Name>John Doe</Name>
      <Phone>(403)555-5678</Phone>
    </record>
  </Source2>
</xmltest>
```

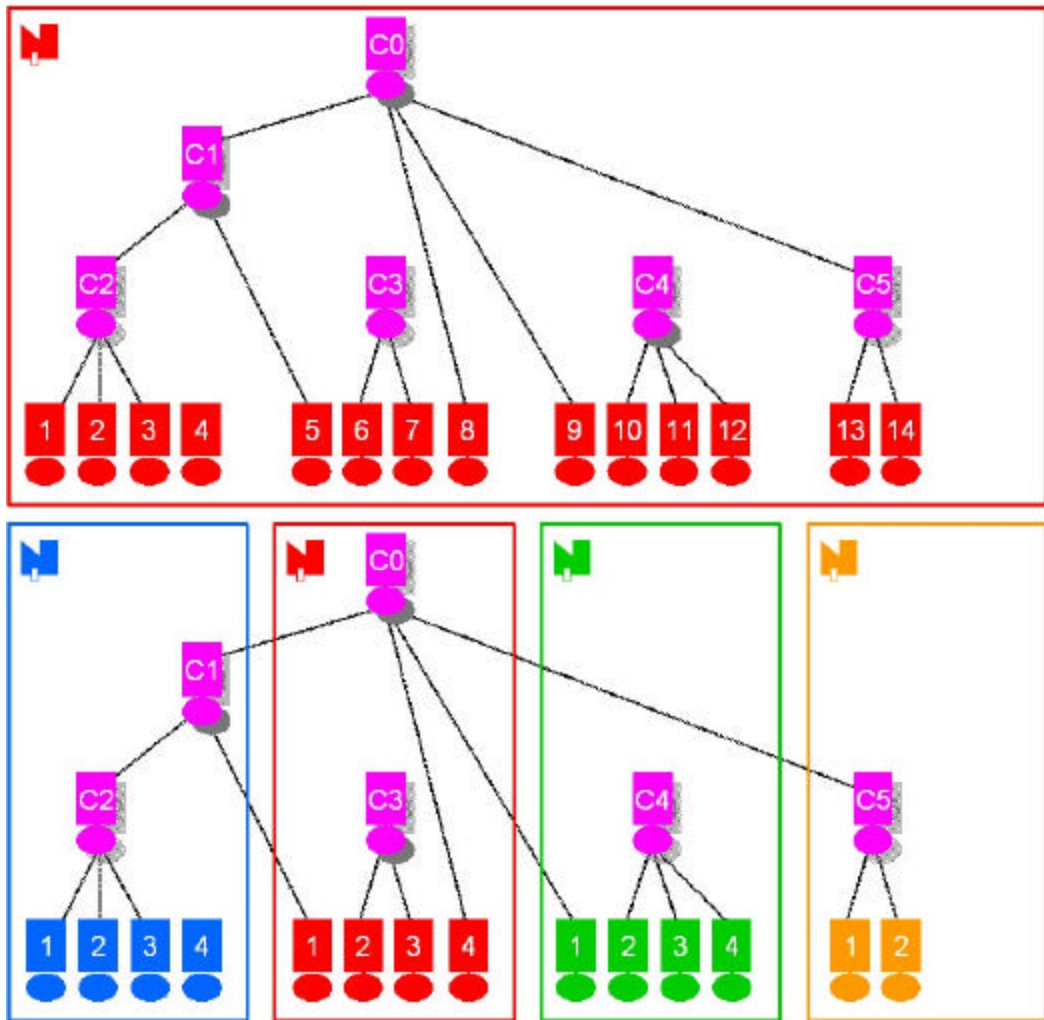


*Agents act on the message in the order in which they are attached.*

## Distribution

An nTegrator application is a distributed hierarchy of collections. Any part of any collection can reside on any Windows computer.

The logical hierarchy of an application is identical whether the application operates on one computer, two, or many.



*nTegrator applications have a 'geographically irrelevant' architecture. In this example, the application architecture is the same whether the system resides on one computer or four.*

## Security

Every nTegrator object is permission protected in ten ways:

- read data
- write data
- read object
- write object
- add Agent
- remove Agent
- read Data Transport persistent data
- write Data Transport persistent data
- read Agent persistent data
- write Agent persistent data



A user's authorities are first matched against collection C-zero's permissions (C-zero being the highest-level collection in the application) to determine the overall authorities that the user has in this application.

These authorities are then passed down the hierarchy and matched against the permissions of each lower-level object.

In this way, the owner of a piece of data that is made available to others can easily determine the degree to which others can see or manipulate that data, and can vary those permissions by user. No matter where it may happen to be in the hierarchy at any given moment, corporate data can only be accessed by authorized people.



In addition, every computer-to-computer message is public key encrypted.

Each copy of nTegrator has its own PKI certificate. A message being sent to computer B from computer A will be encrypted by nTegrator A using nTegrator B's public key. Only nTegrator B can decrypt that message, using its private key.

nTegrator certificates are used only by other instances of nTegrator, so need not be included in a corporation's certificate management process. Encryption and decryption is completely automatic.

This combination of permission-based access to each object and public key encryption between computers gives nTegrator applications a very high degree of security. Each application system becomes, in effect, a virtual private network 'owned' by the application's user and 'approved' by the data's owner(s).

## **Footprint & Cache**

nTegrator has a small footprint: approximately 200K. This allows it to operate unobtrusively in even the smallest Windows device.

The smaller the process and the tighter the code, the less it will be paged in and out of memory by the operating system. As paging involves disk accesses which are many thousands of times slower than memory operations, this attribute can help to optimize performance of nTegrator applications.

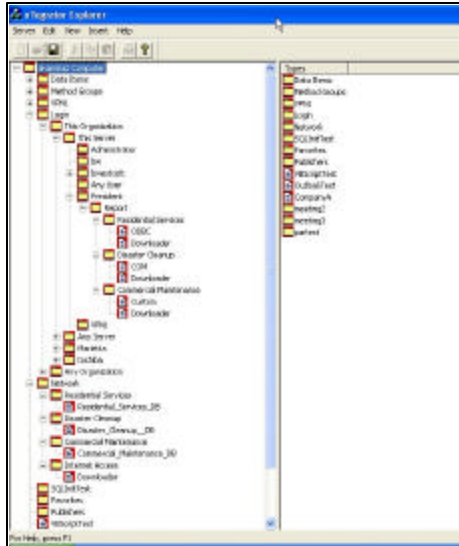
nTegrator objects are maintained in a small database. To speed performance, they are accessed via nTegrator's own in-memory cache. If no external storage device is available, a limited number of objects can be maintained permanently in cache. This allows nTegrator to run on very small devices without external memory.

This caching feature allows application designers tremendous flexibility in how applications are implemented. For example, an application's collection C-zero could reside on a wirelessly-connected PDA. The user could initiate the application from the PDA. The entire application, no matter how large and complex, could operate on 'shore-based' PCs and servers, with the results being displayed in real time on the PDA.

Similarly, an application's collection C-zero could reside on a laptop, with the rest of the application on other shore-based computers. The user could access the primary network over the Internet from anywhere in the world and invoke the application from the laptop. Again, processing could take place wherever appropriate, with results being displayed through the laptop's Internet browser.

nTegrator's compact size and cache-based object management allow application developers free rein to imagine applications never before feasible – and certainly not at anywhere near the low cost of an nTegrator application.

## nTegrator Explorer



nTegrator Explorer is a Windows Explorer-like tool that is used for creating and managing objects and for developing and testing applications.

In Windows Explorer, folders contain other folders and/or files. In nTegrator Explorer, collections contain other collections and/or objects, Data Transports, and Agents.

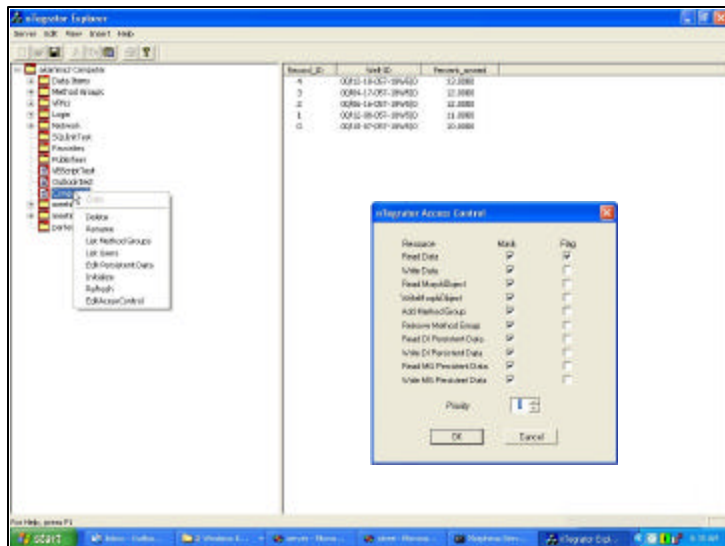
(The biggest difference between the two Explorers is that while Windows Explorer only allows a folder or file to exist once, nTegrator Explorer allows any component to be part of multiple collections.)

nTegrator Explorer is used to create new collections and new objects.

Data Transports and Agents are developed individually using whatever scripting or programming tools are appropriate. Once developed, they are stored in two “home” collections managed by nTegrator Explorer.

A new object is created in a collection when a Data Transport is dragged from its home collection and dropped on to the collection. Agents can then be dropped on to the new object as required.

nTegrator Explorer is also used for housekeeping functions such as adding or modifying permissions associated with each object, testing objects and their components, and reorganizing collections.



## **Application Development**

nTegrator applications can be developed and implemented totally top-down.

Non-trivial business processes can be modeled using, for example, UML (unified modeling language). A variety of tools such as Visio can be used to assist and document this process. Modeling is a technique for documenting business processes.

UML's 'Business Use Cases' can be directly implemented as nTegrator collections. The collections can be decomposed into lower-level collections – a process we call delegation – at the same time as use cases are being analyzed and refined.

When specifying an nTegrator application, the user always starts at the top and works downward, decomposing the main process into component or subordinate processes. Application design exactly parallels this process, to the extent that there is almost no difference in an nTegrator application between the application's specification, its logical design, its physical design, and its construction.

Programming can begin whenever any Agents (processes) have been identified and defined. If lower-level agents have not yet been implemented, XML messages necessary for testing the Agent can be written using any text editor. Each Agent can be developed using the most appropriate tool, independent of how other Agents are produced. As each Agent is, by definition, a small program whose only inputs and outputs are XML messages, it is easily understood by just one programmer. Many years of experience have shown that single-programmer developed programs are produced faster and with more accuracy than those that use multi-person teams.

As Agents and Data Transports are developed, they are maintained in nTegrator Explorer in their 'home' collections. From there, they are dragged and dropped into objects. Each component can be tested individually using XML messages as I/O; complete objects can be tested the same way. nTegrator applications are *developed* as simple, standalone components. These components are *assembled* by dragging and dropping them into objects and low-level collections. The application is *implemented* from the top down by dragging and dropping component collections into high-level collections.

Because of nTegrator's unique geographically independent architecture, an application can be developed and tested on a single computer and then rolled out as a multi-computer distributed application, with no change to the underlying architecture.

## ***NewTech Cleaning Services (nTegrator In Action)***

The following is an illustration of the creation of an integrated solution to a classic business problem using nTegrator.

NewTech Services Inc. is a fictitious cleaning services company that was formed from the merger of three businesses.

It has three divisions:

- Residential Services (house cleaners)
- Disaster Clean-up
- Commercial Maintenance (hospital and factory cleaning)

### **The Need**

The president of NewTech requires a monthly report that contains the following information:

- revenue by division;
- what percentage that revenue is of total revenue ;
- as a comparison, the industry standard percentage of revenue for each category.

<i>NewTech Services Revenue by Division</i>			
<i>month/year</i>	<i>Revenue</i>	<i>% of Total</i>	<i>D&amp;B %</i>
Residential Services	\$\$\$	%	%
Disaster Cleanup	\$\$\$	%	%
Commercial Maintenance	\$\$\$	%	%
<b>Total</b>	<b>\$\$\$</b>	<b>100%</b>	

Because NewTech is a newly formed company, each division maintains its own revenue information in its own accounting system. The three systems use different database structures.

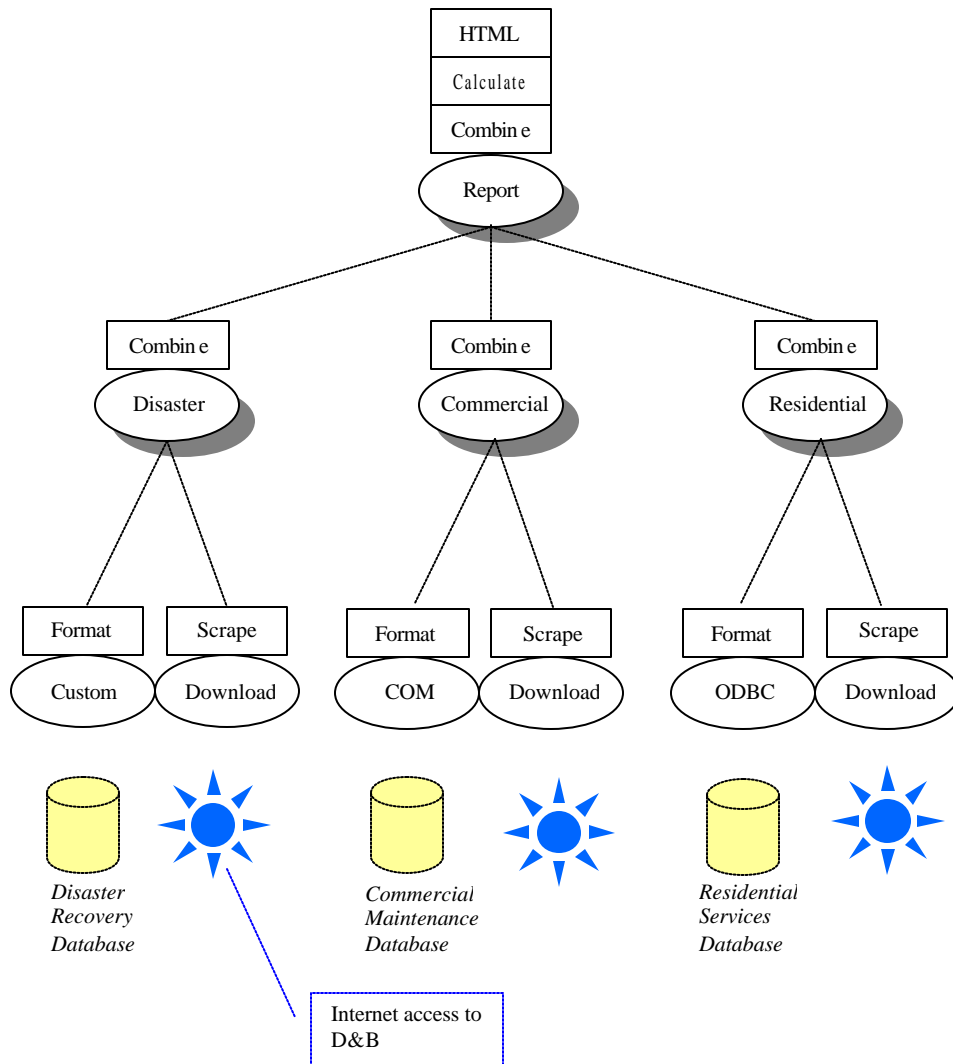
The ‘industry standard’ percentages are assumed to be available on the Dun and Bradstreet (D&B) website.

## The Solution

Using nTegrator to create a solution requires a simple combination of:

- processes that get the data from the various sources; and
- processes that transform the data and perform the necessary calculations.

The logical design of the application is shown below.



For each of the three divisions, there are two Interface Objects. In the diagram these are the six un-shadowed objects. A Data Transport (oval) in each of these

objects either accesses the data from the appropriate accounting system or pulls the D&B data from the D&B website. In both cases, the Data Transport converts the data to XML.

The Residential Services accounting system (lower right) uses a relational database. nTegrator has a standard ODBC Data Transport that is used to pull data from any ODBC-compliant database. The Data Transport labeled “ODBC” in the diagram is one of these standard Data Transports, configured by the user to access the correct data.

The Agent in the same object labeled “Format” is an XSL template that changes the name of the data from its name in the accounting system to the name used by the new application.

The Commercial Maintenance accounting system (lower center) uses a COM-aware database. nTegrator ships with VB Script boilerplate that facilitates development of Data Transports to access COM-aware data sources. The Data Transport labeled “COM” consists of a VB Script Data Transport developed using this boilerplate to access the required data in the accounting system.

The “Format” Agent in the object has the same function as the “Format” Agent in the ODBC object.

The Disaster Cleanup accounting system (lower left) utilizes a proprietary data structure. The Data Transport labeled “Custom” is a VB Script that pulls the required data. Once again, the “Format” Agent has the same re-naming function on the data.

In all three cases, a Data Transport named “Download” accesses the D&B website and pulls down the appropriate page. The attached Agent “Scrape” then extracts the relevant information from the download. The three Data Transports called “Download” are almost-identical C++ programs with one line of code different in each. The three “Scrape” Agents are simple VB Scripts.

The three intermediate-level Collection Objects each have an Agent called “Combine” attached to the Data Transport. This is a standard Agent supplied with nTegrator that combines two or more messages.

Similarly, the high-level Collection Object called “Report” includes the same Agent to combine the three messages received from the subordinate objects.

In addition, “Report” contains an Agent, “Calculate”, which is a VB Script that adds the three revenue amounts together to compute the total revenue, and that calculates what percentage of the total each amount is.

Finally, “Report” has an Agent, “HTML”, which is an XSL template that formats the data into HTML for display using the president’s Internet browser.



*An Introduction*

The entire communication mechanism for the application is XML.

The message that is passed by the Agent “Calculate” to the Agent “HTML” looks like this:

```
<COMBINATION>
  <RESIDENTIAL_SERVICES>
    <AMOUNT> NNNN.NN </AMOUNT>
    <PERCENT> NN.N </PERCENT>
    <DBPERCENT> NN.N </DBPERCENT>
  </RESIDENTIAL_SERVICES>
  <DISASTER_CLEANUP>
    <AMOUNT> NNNN.NN </AMOUNT>
    <PERCENT> NN.N </PERCENT>
    <DBPERCENT> NN.N </DBPERCENT>
  </DISASTER_CLEANUP>
  <COMMERCIAL_MAINTENANCE>
    <AMOUNT> NNNN.NN </AMOUNT>
    <PERCENT> NN.N </PERCENT>
    <DBPERCENT> NN.N </DBPERCENT>
  </COMMERCIAL_MAINTENANCE>
  <TOTAL> NNNN.NN </TOTAL>
</COMBINATION>
```

This is the message that “HTML” formats for display as the required report. (The date is simply drawn from the president’s computer.)

“Calculate” receives from “Combine” this message:

```
<COMBINATION>
  <RESIDENTIAL_SERVICES>
    <AMOUNT> NNNN.NN </AMOUNT>
    <DBPERCENT> NN.N </DBPERCENT>
  </RESIDENTIAL_SERVICES>
  <DISASTER_CLEANUP>
    <AMOUNT> NNNN.NN </AMOUNT>
    <DBPERCENT> NN.N </DBPERCENT>
  </DISASTER_CLEANUP>
  <COMMERCIAL_MAINTENANCE>
    <AMOUNT> NNNN.NN </AMOUNT>
    <DBPERCENT> NN.N </DBPERCENT>
  </COMMERCIAL_MAINTENANCE>
</COMBINATION>
```

Note that the percent and total amounts are missing. These are the message portions that are calculated by “Calculate”.



## An Introduction

The three messages received by the Data Transport “Report” are similar to each other. The one from the “Residential” collection looks like this:

```
<COMBINATION>
  <RESIDENTIAL_SERVICES>
    <AMOUNT> NNNN.NN </AMOUNT>
    <DBPERCENT> NN.N </DBPERCENT>
  </RESIDENTIAL_SERVICES>
</COMBINATION>
```

Similarly, the two messages combined into one by “Residential” look like this:

```
<RESIDENTIAL_SERVICES>
  <AMOUNT> NNNN.NN </AMOUNT>
</RESIDENTIAL_SERVICES>
```

and

```
<RESIDENTIAL_SERVICES>
  <DBPERCENT> NN.N </DBPERCENT>
</RESIDENTIAL_SERVICES>
```

Finally, the message generated by the “ODBC” Data Transport is:

```
<RESIDENTIAL_SERVICES>
  <MONTH-TOTAL>
  NNNN.NN
  </MONTH_TOTAL>
</RESIDENTIAL_SERVICES>
```

which the attached Agent “Format” changes to:

```
<RESIDENTIAL_SERVICES>
  <AMOUNT> NNNN.NN </AMOUNT>
  <DBPERCENT> NN.N </DBPERCENT>
</RESIDENTIAL_SERVICES>
```

All other messages in this application are similar to those just described.

### **Physical Implementation**

The application is physically installed on five computers, each of which runs a copy of nTegrator:

- The president’s computer contains collection C-zero: “Report” and the three combining collections “Disaster”, “Commercial”, and “Residential”. [All objects have the same names as their Data Transports.]

# **nTEGRATOR**

## *An Introduction*

- The computer that hosts the Residential Services accounting system contains the Interface Object “ODBC”. (Or, alternately, the object resides on a computer that can issue an ODBC call to the accounting system.)
- The computer containing the Commercial Maintenance accounting system hosts the Interface Object “COM”.
- The computer containing the Disaster Recovery accounting system hosts the Interface Object “Custom”.
- Whichever computer is used to access the Internet -- likely the president’s computer – hosts the three “Download” Interface Objects.

Final implementation is almost identical to the logical design since nTegrator solutions are truly top-down and the delineation between logical and physical layers is almost non-existent.

The lines of code required for this application are:

<u>Agent / Data Transport</u>	<u>Lines of code</u>
“Calculate” Agent: VBScript	50
“COM” DT to access DB using COM: VBScript	70
“Custom” DT to access proprietary DB: VBScript	150
“Scraper” Agent 1: VBScript	20
“Scraper” Agent 2: VBScript	20
“Scraper” Agent 3: VBScript	20
“HTML” Report format Agent: XSL template	50
“Format” Agent to change data name 1: XSL template	19
“Format” Agent to change data name 2: XSL template	19
“Format” Agent to change data name 3: XSL template	19
“Download” DT to access D&B website 1: C++	200
“Download” DT to access D&B website 2: C++	1
“Download” DT to access D&B website 3: C++	1
“Combine” Agents to combine messages	0
“ODBC” DT to access SQLDB	1

# **nTEGRATOR**

## *An Introduction*

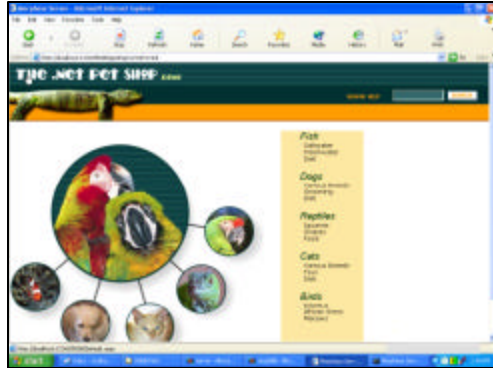
Total code	640
<i>Boilerplate (80%)</i>	<i>500</i>
<b>Total original code</b>	<b>140</b>

This is a good example of how nTegrator allows systems to be built using data that is in different formats on different computers.

It is also a good example of nTegrator's "programmer-assisted end-user" philosophy. Only 140 original lines of code had to be developed, and, arguably, a good proportion of that could be done by a technically knowledgeable user. The code was used to develop components. Most of the code was boilerplate that would have been copied from previous components or from standard components. Using nTegrator Explorer, any competent end user could have dragged and dropped those components into a working application.

## Comparative Efficiency

Benchmarks for the Pet Shop demo developed by Sun, emulated by Microsoft, and then implemented using nTegrator, indicate that nTegrator offers significant savings in development effort:



for user interface, 11% fewer lines of code than .NET and 72% fewer than J2EE;

for business logic, 53% fewer lines of code than .NET and 93% less than J2EE;

for database processing, 89% fewer lines than .NET and 81% less than J2EE.

Fewer lines of code means faster, cheaper implementation.



*An Introduction*

## ***Product Status***

nTegrator is presently in beta test mode.

Beta testers are solicited.

If interested, you should contact nBoundary Software Inc. at [cbuchanan@nboundary.com](mailto:cbuchanan@nboundary.com).

## **Programmer's Note**

*Even though Interface Objects and Collection Objects have been described as having differing functions and characteristics, they are actually implemented identically in nTegrator. The nTegrator engine neither knows nor cares which type of object it is instantiating at any time.*

*Even though Data Transports have been described differently for Collection Objects and for Interface Objects, all Data Transports are, in fact, identical in concept.*

*Internally, Data Transports are called Data Items.*

*Similarly, all types of Agents are identical as far as their implementation and handling within nTegrator are concerned. Some Agents such as those that combine messages would only occur in Collection Objects, but they are not in any way different in concept from any other Agents.*

*Internally, Agents are called Method Groups.*

*XML messages do not necessarily “move” from Agent to Agent, even though it is helpful to think of them that way. In fact, Agents and Data Transports are subroutines that are called by their host object; the message that is processed may simply have its address passed from routine to routine.*

*When an XML message is transmitted across a network, it actually enters the target computer via a high-level collection on that computer and is then ‘passed’ to the target object. This is transparent to both the user and the application’s designer.*

*Within an nTegrator application, XML messages are maintained in ‘compiled’ form for more efficient processing. For simplicity of understanding, it is practical to think of them in the quasi-English form in which they appear in readable scripts.*

*No PKI certificate management is required. As each copy of nTegrator has its own certificate, and as nobody else uses these certificates except for other copies of nTegrator, there is no need to include them in corporate certificate management.*

*In the Pet Shop example, counts of lines of code for J2EE and .NET were furnished by Microsoft. Sun would probably claim much of their code is boilerplate, so the effective number of new lines of code should be less for their implementation. If that were taken into account, nTegrator’s code count would drop similarly, as much of nTegrator’s code is boilerplate. The ratios would remain similar to what is presented in this document.*