An Esri® White Paper • August 2009

ArcGIS® Server in Practice Series:
Best Practices for Creating an ArcGIS Server Web Mapping Application for Municipal/Local Government

Esri, 380 New York St., Redlands, CA 92373-8100 USA
TEL 909-793-2853 • FAX 909-793-5953 • E-MAIL info@esri.com • WEB www.esri.com

Copyright © 2009 Esri
All rights reserved.
Printed in the United States of America.

The information contained in this document is the exclusive property of Esri. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by Esri. All requests should be sent to Attention: Contracts and Legal Services Manager, Esri, 380 New York Street, Redlands, CA 92373-8100 USA.

The information contained in this document is subject to change without notice.

Esri, the Esri globe logo, ArcGIS, ArcSDE, ArcMap, ArcInfo, ModelBuilder, ADF, www.esri.com, and @esri.com are trademarks, registered trademarks, or service marks of Esri in the United States, the European Community, or certain other jurisdictions. Other companies and products mentioned herein may be trademarks or registered trademarks of their respective trademark owners.

ArcGIS Server in Practice Series: *Best Practices for Creating an ArcGIS Server Web Mapping Application for Municipal/Local Government*

An Esri White Paper

Contents **Page**

About the ArcGIS Server in Practice Series 1

Introduction 1

Use Case Requirements 2
- User Workflow 2

GIS Datasets 4

Solution Architecture 4
- Server Configuration 4
  - Physical Hardware 5
- Data Publication Planning 5
- Web Services 7
  - Riverside Street Map Map Service 8
  - Imagery Basemap Map Service 15
  - Utilities Map Service 16
  - U.S. Street Geocoding Service 19
  - Geometry: Map and Geometry Service 21
- Web Application Development 24

ArcGIS Server Solution Assessment 26

Summary 28

Contents **Page**

Appendixes

Appendix A: Riverside Viewer Web Mapping Application and User Workflow 29

Appendix B: GIS Datasets for the City of Riverside, California 40

Appendix C: Riverside Street Map—Scale Cache Levels 42

Appendix D: The Geoprocessing Service 47

Appendix E: Map Queries and the Geometry Service 50

Appendix F: Testing Methodology and Definitions 56

ArcGIS Server in Practice Series: *Best Practices for Creating an ArcGIS Server Web Mapping Application for Municipal/Local Government*

About the ArcGIS Server in Practice Series

The ArcGIS Server in Practice series provides practical information for the configuration and implementation of ArcGIS® Server solutions. Documents in this series explore common and well-defined user workflows and system configurations. They provide example use cases for describing ArcGIS Server best practices. The emphasis of this series is to examine use cases from a holistic point of view and provide useful information enabling users to understand how to optimally configure and implement ArcGIS Server for real-world applications.

Introduction

The scenario in this document is for a medium-sized city with a population of approximately 300,000 and a geographic area of approximately 80 square miles that wants to build a Web mapping application. It combines geographic information, such as cadastral maps, utility networks, and imagery, from different municipal/local government departments. The intent is to provide an intuitive Web browser-based application for performing common spatial analysis tasks (locating addresses, creating proximity reports, inspecting the status of different assets, etc.) that is accessible to all city government personnel.

This document guides users through the process of building such a Web mapping application and its supporting Web services. The primary objective is to illustrate best practices and focus on some of the key decisions that were made to effectively support the business requirements of the scenario described above. To keep the document within a limited scope, a typical ArcGIS Server Standard Workgroup deployment is used.[1] This deployment configuration also helps achieve the secondary objective of providing some practical information on the support capacity (e.g., number of supported users) for scalability of a typical ArcGIS Server Workgroup system.

This document covers the following:

- Use case requirements
- GIS datasets

[1] ArcGIS Server Workgroup deployments are limited to a single machine hosting the Web server, GIS application server, and database server tiers.

- Solution architecture
  - Server configuration
  - Data publication planning
  - Web services
  - Web application development
- ArcGIS Server solution assessment

Use Case Requirements describes the general Web mapping application and the typical user workflow interaction with it. GIS Datasets provides an overview of the geographic information system (GIS) data used in the Web application. The Solution Architecture section is divided into several subsections: Server Configuration, Data Publication Planning, Web Services, and Web Application Development. Each subsection discusses the setup, development, and implementation of the major components of the ArcGIS Server solution for the scenario. Throughout the Solution Architecture section, there is an emphasis on the reasoning behind implementation decisions, the efficient design of Web services, and the optimization of the Web application for scalability. In the ArcGIS Server Solution Assessment section, a general assessment of the overall system performance is discussed. The document concludes with a summary of the ArcGIS Server solution provided for the scenario.

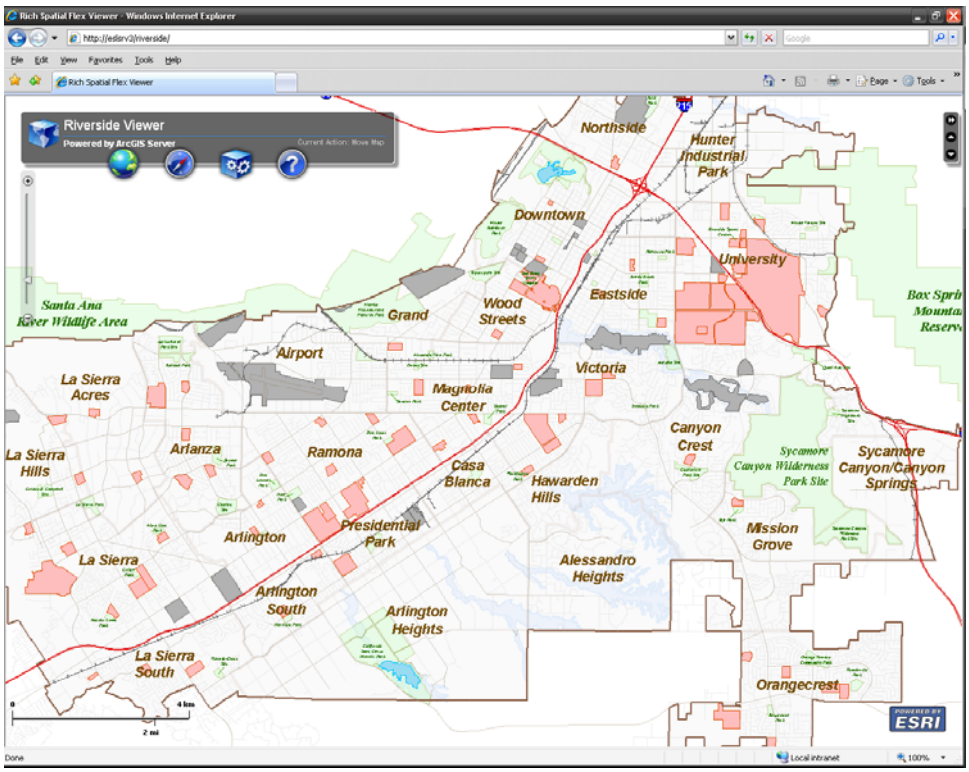
The target audiences for this document are beginning to intermediate users with an understanding of fundamental ArcGIS Server concepts. Note that some of the functionality implemented in the final Web mapping application (such as in the reporting task appendixes) is more appropriate for advanced users.

Use Case Requirements

User Workflow

The Web mapping application is named Riverside Viewer. It is a simple, easy-to-use map viewer that includes some basic spatial analysis functions commonly found in municipal/local government workflows, including mapping, geocoding, search capabilities, and reporting. The Web application will be used by all city government personnel—approximately 60 people with varying levels of GIS analysis skills—so its design needs to be simple and intuitive (see figure 1).

Figure 1
Riverside Viewer Web Mapping Application—Default Appearance when Initially Loaded in a Web Browser



The Web application controls are located in the upper left corner. Drop-down menus appear when the cursor is paused on one of the four control icons: Map, Navigation, Tools, and Help. There is also a map view scale selector bar. Additional control windows may appear in the display (typically in the upper right corner) depending on the functionality selected. An example user workflow for the Web application is described in table 1.

Table 1
Example Web Application User Workflow

Step	Action
1	Start the application at its default extent: overview of the city of Riverside with the street map displayed.
2	Geocode an address (e.g., 3090 Rice Road); center the map on the candidate result.
3	Switch the basemap to imagery, then switch back to street map.
4	Retrieve further details on the parcel at the address by identifying it.
5	Find a parcel by its Assessor Property Number (APN) (e.g., 218181018) and zoom to it.

Step	Action
6	Create a report of all properties within 300 feet of the parcel.
7	Find a school by its name (e.g., Riverside City College) and zoom to it.
8	Turn on the utilities dynamic layer.
9	Turn off the meters layer and refresh the map display.
10	Find a parcel by its APN and zoom to it.
11	Turn on the meters layer.
12	Find an electrical pole by its SynergenID and zoom to it.
13	Zoom to the full extent and close the application.

For more details on the Riverside Viewer Web mapping application and screen captures of the example user workflow, see appendix A.

GIS Datasets

GIS data for the city of Riverside, California, was used in this scenario. The city of Riverside has a population of 311,575 people and covers an area of approximately 80 square miles. Its GIS database contains over 60 different types of data including vector, imagery, utility networks, and annotation data, totaling 12.5 GB. A high-level summary of the GIS data content used for the Web mapping application is listed in table 2. For further details, see appendix B.

Table 2
GIS Datasets for the City of Riverside, California

Dataset Name	Dataset Type	Size
Addresses	Point	84,213 records
Schools	Polygon	87 records
Buildings	Polygon	105,562 records
Electric poles	Point	4,559 records
Parcels	Polygon	79,663 records
Aerial imagery	Raster	59 MrSID files
Various text	Annotation	Over 100,000 records

Solution Architecture

This section of the document discusses the hardware setup, planning, and implementation of the various components of the ArcGIS Server solution for this municipal/local government scenario.

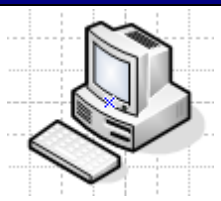
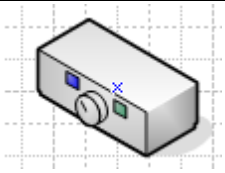
Server Configuration

As mentioned previously, a typical ArcGIS Server Standard Workgroup deployment is used for this scenario—in other words, a single four-core server machine deployment of ArcGIS Server that includes the Web server, server object manager (SOM), server object container (SOC), and database tier. Two geodatabases were used to support the Web application's various services: a Workgroup ArcSDE® geodatabase contained the electrical network data, and a file geodatabase held the remainder of the GIS datasets (for details, see appendix B).

Physical Hardware

Two machines were used for this use case. One represents the publication server and hosts the entire ArcGIS Server deployment. The other represents the client and was used to simulate user load, as well as monitor different key indicators of the ArcGIS Server instance.[2] Both are connected on a network with a 1,000-megabit throughput capacity. Table 3 summarizes the physical hardware of the ArcGIS Server solution.

Table 3
General Information on the Physical Hardware

	Hardware	Software
 Client	**CPU:** 1 Intel XEON, 3.6 GHz **Memory:** 2 GB **Disk:** SATA, 120 GB, 10K rpm	Windows XP Visual Studio 2008 Test Team
 Network	1,000 megabit	Not applicable
Server	**CPU:** 2 Intel XEON, 3.0 GHz Dual Core **Memory:** 16 GB **Disk:** Dual SATA, 146 GB, 10K rpm **RAID:** 1	Windows 2003 SQL Server Express 2008 ArcGIS Server Workgroup

Data Publication Planning

One of the first steps in designing an ArcGIS Server solution is to carefully examine and assess the purpose(s) of the GIS data that will be used in the Web mapping application. The GIS data should be classified into two groups: operational data and basemap data.[3] Operational data will be actively used in the Web application, for example, feature classes that will be edited or queried by Web users. Basemap data will primarily be used as referential background in the Web application. In other words, basemap data is present within the application to support the operational data.

Properly assigning GIS data into the two groups is important, because it affects the usability of the Web application and its performance. In general, basemap data should be implemented as cached map services, because the data will change either infrequently or not at all. In contrast, operational data may need to be implemented as dynamic map services because users might require additional capabilities (the ability to toggle between

[2] For more information on how the ArcGIS Server system was monitored, see appendix F.

[3] The two group terms are also sometimes referred to as operational layers and base layers, respectively, in ArcGIS Server documentation, but they have the same overall meaning. More information can be found in the ArcGIS Desktop Help topic Steps for implementing GIS map applications.

data layers, dynamic labeling, etc.) in the Web application. In addition, the operational data may change too frequently to achieve effective cache updates.

A good Web application design strategy is to minimize the amount of information shown via dynamic map services. Their use involves live queries applied to the geodatabase, potentially adding extra overhead to the ArcGIS Server solution. It is strongly recommended that users cache information as much as possible in the Web application to optimize the use of the hardware resources.

Tip 1: Use cached map services for serving GIS data with ArcGIS Server whenever possible. They yield better performance and put less load on the server compared to dynamic map services.

For this scenario, most of the city of Riverside, California, GIS data was implemented as cached map services to optimize performance in the Riverside Viewer Web mapping application. The exception is the electrical network data (referenced by the Utilities map), which is used as operational data in the Web application. Details about individual services are described later in the document. Three map documents (i.e., Street, Aerial Imagery, and Utilities) are used in the Web application and shown in figures 2 through 4; they are the published resources behind the ArcGIS Server map services.

Figure 2
Street Map at 1:80,000 Scale

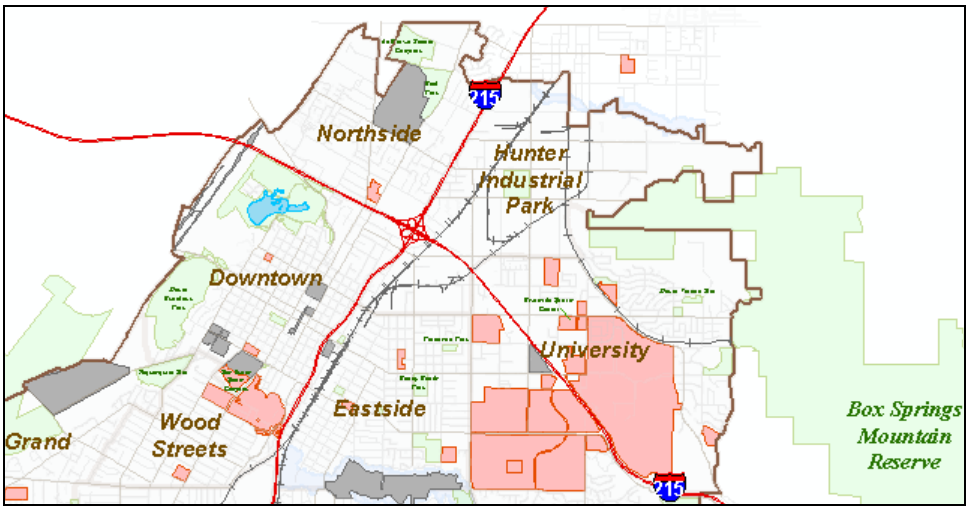
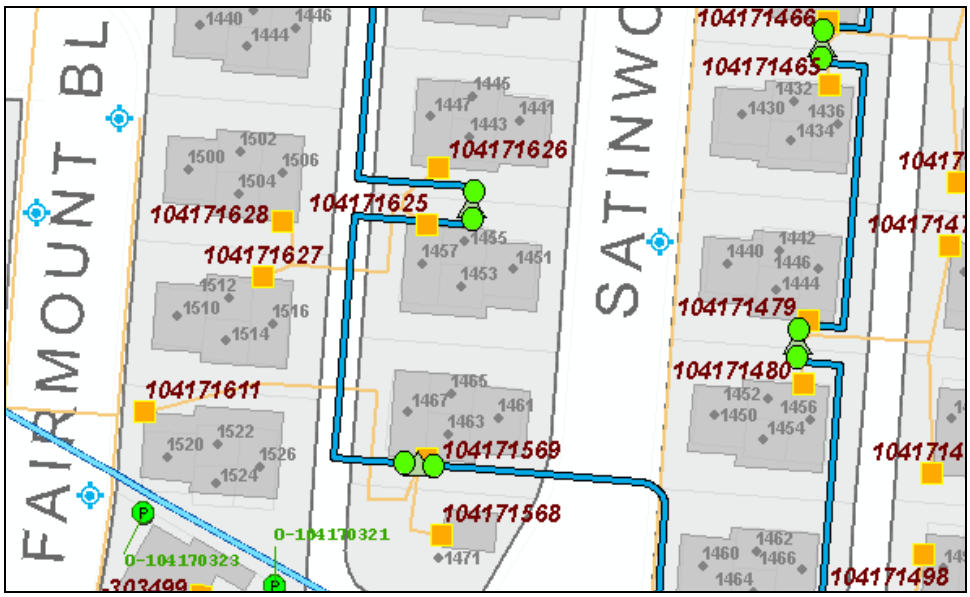


Figure 3
Aerial Imagery Map at 1:2,000 Scale

Figure 4
Utilities over Street Map at 1:1,000 Scale



Web Services

Several GIS Web services were created to satisfy the mapping and querying requirements of the example user workflow described earlier. Based on the available GIS datasets, five different Web services were created to support the Web mapping application. Three of them are map services, one is a geocoding service, and one is a geometry service (see table 4). Each Web service is described in detail, including some discussions explaining why certain configuration properties were implemented, and some best practices are highlighted to help optimize the ArcGIS Server system.

Table 4
List of Web Services

Web Service Name	Type of Service	Purpose
Riverside Street Map	Map (cached)	Main basemap in the application and queries to Parcels feature class
Imagery Basemap	Map (cached)	Provides background imagery
Utilities	Map (optimized dynamic)	Shows detailed electrical information and queries to elements in the geometric network
U.S. Street	Geocoding	Finds addresses
Geometry	Geometry	Supports report generation functionality

Riverside Street Map Map Service

This map service is the main basemap for the Riverside Viewer Web mapping application. It is used for referencing information such as neighborhoods, transportation routes, and city features (e.g., parks, schools, and other landmarks) at large scales. It also provides detailed information on streets, parcels, and house numbers at smaller, more detailed scales.

Several key datasets for the city of Riverside were collected and symbolized as a cartographic basemap (named Riverside_BaseMap) in ArcMap™. Best practices for designing maps for optimal performance were followed.[4] Examples include optimizing the map content, map symbols, and text and labels. For Riverside_BaseMap, annotations were used extensively instead of dynamic text. This accelerates display response times and ensures the best text placement possible. In terms of display performance, on average, refreshing the map in ArcMap was accomplished in the subsecond range. Refreshing at larger scales, such as 1:500 and 1:2,000, yielded faster response times—approximately 0.2 seconds—over smaller scales, like 1:25,000, which took approximately 1 second.

Even though Riverside_BaseMap was designed for optimal performance, when the map service was created, it was cached[5] (instead of being a dynamic map), because the information it contained is relatively static. The cartographic optimizations helped reduce the amount of time needed to create and update the cache tiles. In general, using cached map services helps increase the capacity of the server under heavy loads.

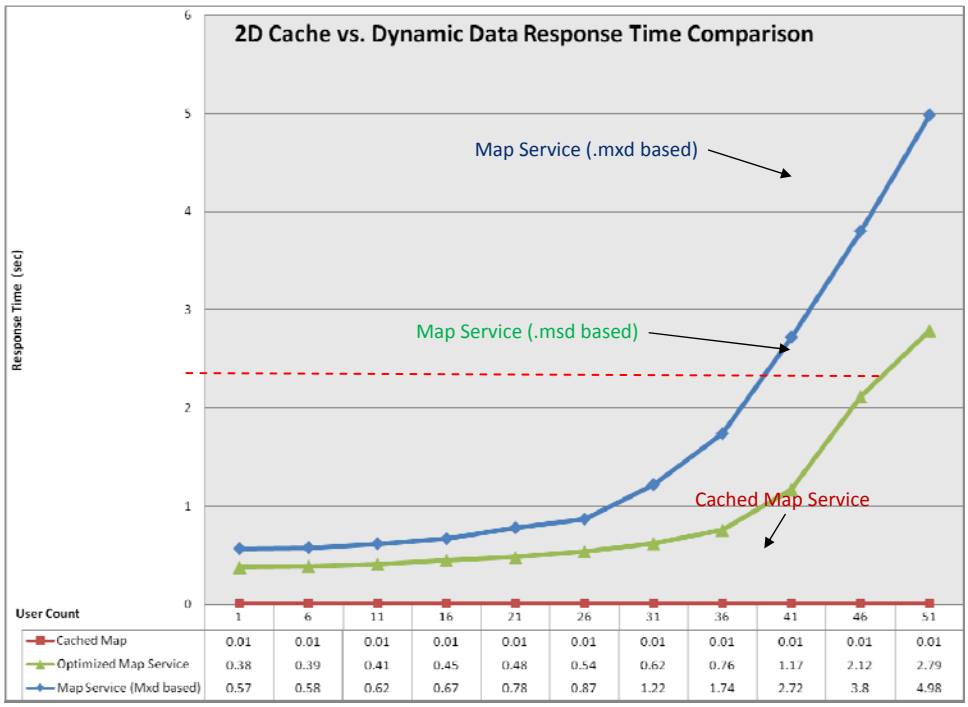
[4] More information can be found in the ArcGIS Desktop Help topic An overview of designing maps for optimal performance.

[5] More information on map caching can be found in the ArcGIS Server Help topic What is map caching?

To verify and confirm that caching is a better option than using a dynamic map service in this specific case, a small test framework was set up. The test compared the performance of both choices (i.e., cached versus dynamic). A user[6] load was simulated and applied against the two different types of services by defining a simple map navigation workflow (e.g., panning and zooming) with a six-second think time between map interactions. Figure 5 summarizes the test framework results. It compares the response times of the Riverside_BaseMap map services (cached in red versus dynamic in blue) as the number of users accessing the service increases.

Notice that the chart also contains a third test result (green line). Two different versions of the Riverside_BaseMap dynamic map service were tested. One was based on an ArcMap document (.mxd file), and the other was based on a map service definition (.msd) file. ArcGIS Server 9.3 can only create dynamic map services using .mxd files, whereas ArcGIS Server 9.3.1 introduces the concept of *optimized map services*, configured with a new type of file (.msd files) for dynamic map services. Optimized map services will be discussed later in the document.[7]

Figure 5
Test Framework Summary—Cached Map Service vs. Dynamic Map Service



User Count	1	6	11	16	21	26	31	36	41	46	51
Cached Map	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Optimized Map Service	0.38	0.39	0.41	0.45	0.48	0.54	0.62	0.76	1.17	2.12	2.79
Map Service (Mxd based)	0.57	0.58	0.62	0.67	0.78	0.87	1.22	1.74	2.72	3.8	4.98

[6] In this document, a *user* is defined as an intelligent consumer of data services within an Internet application requiring a pause between interactions to decide if further interaction is needed. The pause between user interactions is described as *think time*.

[7] A more detailed discussion on optimized map services can be found under the Utilities Map Service subsection in this document (see page 18).

A threshold response time of two seconds was set, which is the maximum response time considered to be acceptable for this basemap. Observe that when the dynamic map service based on the .mxd file was accessed by 36 or more users, its response time increased dramatically beyond the two-second threshold. The optimized map service (based on the .msd file) yielded better throughput, enabling support for 10 additional users (bringing the total to 46 users) under the response time threshold. Conversely, the cached map service response time remained constant throughout the test, despite the increase in users. It could serve 50 or more concurrent users with a consistent response time of less than half a second. While not shown in figure 5, up to 150 concurrent users were simulated in the test framework, and the response time was still in the subsecond range. The results from the test framework confirm that a cached map service is the best option for publishing the Riverside_BaseMap map document.

It is important to note that since cached maps do not generate map images at run time (assuming on-demand caching is not being used), using a cached map enables the ArcGIS Server administrator to reduce the number of map instances on the server. This, in turn, frees up memory consumption in the SOC. Since cached map services typically use minimal CPU resources, they allow other services to take advantage of the available server hardware resources, even under heavy load.

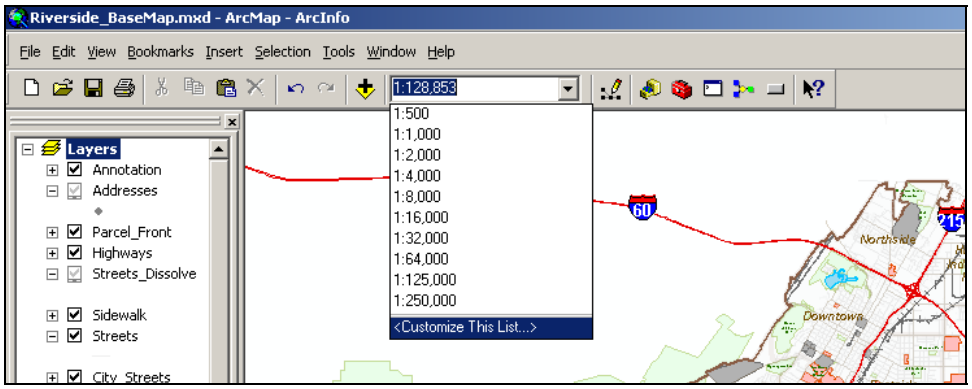
Special considerations must be taken into account when authoring a map document that will be cached.[8] In a dynamic map service, the end user can zoom to any scale. In a cached map service, the number of scales is predefined. The approach for defining the preset map scales for a cached map service is to determine which map scales are needed by the business requirements of the Web mapping application. It is a good idea to take this into account early in the design process when creating cached map services for ArcGIS Server.

One of the Web application's requirements is that it must display information from a citywide level down to 1:500, the scale where detailed information for streets and parcels can be displayed. A strategy for defining map scales is to double the minimum map scale incrementally until the maximum map scale needed is reached, for example, 1:500 → 1:1,000 → 1:2,000 → 1:4,000, and so on, up to the citywide scale, approximately 1:250,000.[9] Since the Riverside_BaseMap map document will only be used at the scales defined in the map cache schema, it was authored in ArcMap at the specific scale cache levels. The intent is to ensure that the map will display at its best for each scale in the cache (see figure 6). Appendix C contains screen shots of the different scale cache levels for Riverside_BaseMap.

[8] More information on map caching requirements can be found in the ArcGIS Server Help topic Tips and best practices for map caches > Preparing the map document.

[9] The maximum map scale (i.e., citywide level) for the city of Riverside was determined by viewing the citywide boundary of the data in ArcMap and noting the map scale displayed.

Figure 6
Editing the Map Document at the Specific Cache Levels



The following scale cache levels for the Riverside_BaseMap map service were used:

- 1:250,000
- 1:125,000
- 1:64,000
- 1:32,000
- 1:16,000
- 1:8,000
- 1:4,000
- 1:2,000
- 1:1,000
- 1:500

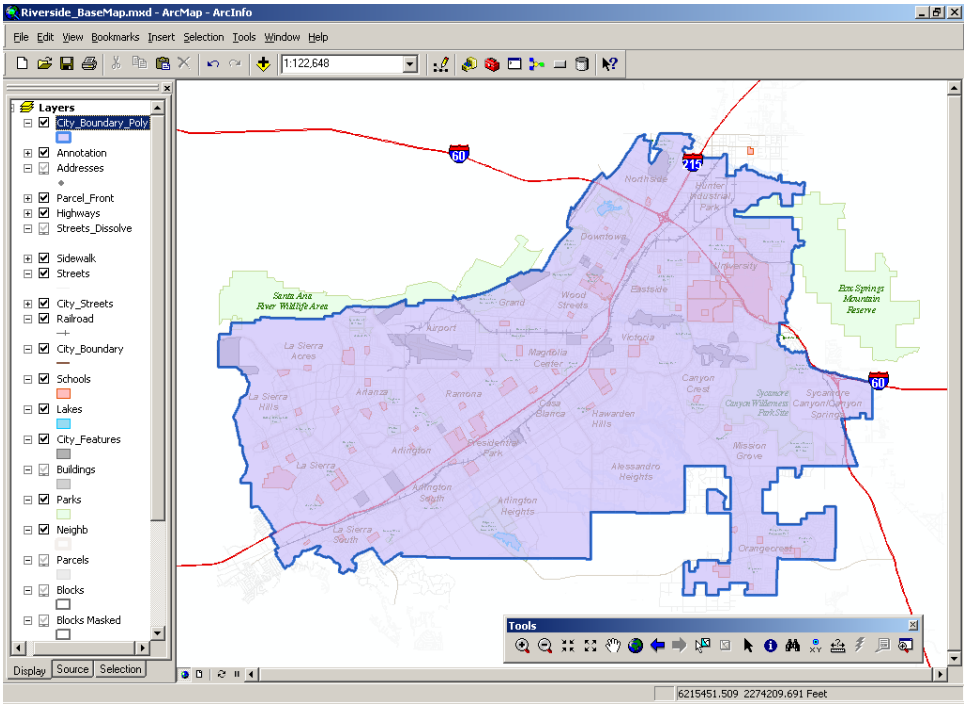
Caching the entire Riverside_BaseMap map took approximately 18 minutes for this ArcGIS Server environment (see earlier sections for configuration specifics). This is a relatively short amount of time to build a cache, which allows frequent and easy cache updates. Map server instances for the Riverside Street Map map service were changed from two to four, because the ArcGIS Server environment had four cores. Map caching is very CPU intensive and generally does not benefit from having much more than one map instance per CPU core while generating the cache. The actual number that generates the most tiles per hour is $5n/4$, where n is the number of CPU cores (e.g., 5 instances for a four-core machine, 10 instances for an eight-core machine). If a caching system configured with $5n/4$ instances appears to be underutilizing CPU resources, it is likely due to disk or network I/O bottlenecks.

Note that the 18 minutes used to create the map cache is the time it took to cache *all* the geography within the full extent of the map and at all cache levels.

Tip 2: In general, avoid caching areas in the map that will not be used. This will not only save disk space, but more importantly, it will help expedite the map cache building process (i.e., make it much faster).

The highlighted blue area in the map in figure 7 shows the area of interest for the Riverside Viewer Web mapping application. It is anticipated that users of the Web application will only navigate the map within this area and not beyond the city limits.

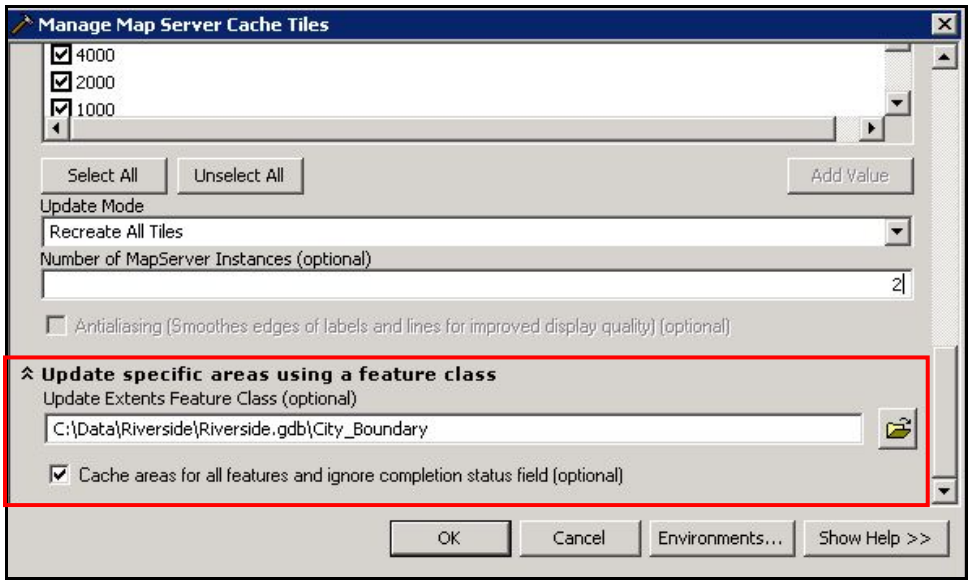
Figure 7
City Boundary for the City of Riverside, California



Observe that approximately half of the map area (at its full extent) lies outside the city. Therefore, the city's boundary can be used to constrain the spatial extent of the map caching process. This is achieved by inputting the city of Riverside boundary feature class into the Manage Map Server Cache Tiles geoprocessing tool[10] as shown in figure 8.

[10] More information on the Manage Map Server Cache Tiles tool can be found in the ArcGIS Desktop Help topic Manage Map Server Cache Tiles (Server).

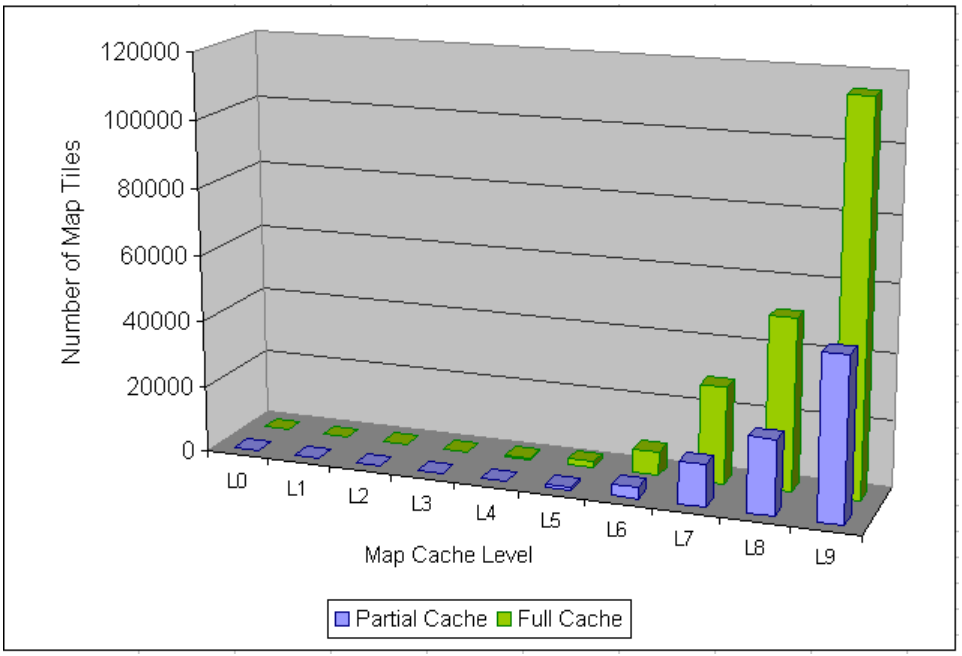
Figure 8
Manage Map Server Cache Tiles Geoprocessing Tool Dialog Box



To validate the theory that caching a subset of the map (i.e., the city boundary extent) versus the entire map is optimal, another small test was performed. Two different map caches were generated for the Riverside_BaseMap map service—one cache of the city boundary extent and another cache of the entire map extent.

Figure 9 compares the number of tiles created when caching the entire map (green bars) versus building a partial cache of the map that is relevant to the city (purple bars). The number of tiles is the same at small scales (e.g., levels 0, 1, 2, 3, 4), but the partial map cache is significantly smaller at higher levels. When a cache is built for the entire map, essentially map tiles at 1:500 scale are generated for areas that are outside the city boundary and do not show any relevant data for the Riverside Viewer Web application.

Figure 9
Comparison between Cache for the Entire Map vs. Cache for the City Boundary



For this specific case, the number of map tiles was reduced from over 200,000 tiles to approximately 90,000. The advantages of reducing the number of tiles in the cache are less use of disk space and shortened processing time when creating the map cache. The time to create the map cache was reduced by almost 50 percent, from approximately 18 minutes (full cache) to 10 minutes (partial cache).

When building a partial cache, on-demand caching can be used for areas in the map where a cache has not been created; it ensures that map tiles will be created at run time if a user visits that area. An alternative option is to display a *Data not available* tile for empty cache areas. In some cases, configuring the service to return a *Data not available* tile may yield a better user experience than returning a blank map display.[11] The *Data not available* option was selected for this cache.

Tip 3: Esri recommends being conservative when building partial caches. Users should benefit performance-wise from the map cache in most instances and trigger on-demand caching rarely or never. When visiting areas that have not been cached, users will suddenly notice that the map navigation becomes slightly sluggish, as the map tiles are created on the fly.

[11] More information on selecting the appropriate option for partial caches can be found in the ArcGIS Server Help topic Common caching questions.

Recall that map cache images can use one of two image format types: JPEG or PNG. The former is ideal for raster-based basemaps with a lot of color variation, such as imagery, while the latter should be used for basemaps with a few simple colors. PNG is the map tile image format selected for this service. A small holistic trial and error test between PNG 8 versus PNG 32 was performed to determine the ideal bit depth per channel. A representative map tile of an urban downtown housing area in the map was sampled; PNG 8 used 16 KB of storage, while PNG 32 used 32 KB of storage. PNG 8 was selected because it yielded quality images and the smallest map tile size.[12]

One administrative side note: Whenever the underlying data changes for a cache, the cache will need to be updated. The process of updating map caches can be automated using ArcGIS geoprocessing tools. When the cache update runs, the server will mostly be dedicated to creating the cache tiles.[13] In this case, since the ArcGIS Server system is on a single machine, the cache update can be scheduled to occur at night or during the weekend (i.e., during a time period where few or no users are accessing the Web application). Alternatively, a second machine can be configured for the purpose of updating the cache. Typically, this extra machine would run on a staging deployment to ensure uninterrupted service.

Imagery Basemap Map Service

This data is also used for referencing information in the Riverside Viewer Web mapping application. However, it is published separately from the Street Map service to enable users to switch between the street map and image data. By building two separate map services (i.e., Street Map and Imagery Basemap), Web application users can easily toggle between the two reference data basemaps. The Imagery Basemap map service could also be used as a backdrop for other Web applications.

Another advantage of this implementation approach is the optimization of the map cache format. If data from both the Street Map and Imagery Basemap services was combined, a PNG 32 map tile image format would be needed. This would generate larger image map tiles. Alternatively, if a JPEG map tile image format was selected, the vector and annotation data would likely not be of high enough quality for use in the Web application.

Tip 4: Imagery map caches should be separate from vector data map caches. Do not combine imagery and text data within the same cache; instead, create one cache for imagery and a separate cache for text (e.g., annotation).

To optimize performance, the imagery was cached following guidelines similar to those described for the Riverside Street Map map service—in other words, using the same scale cache levels, only caching within the city boundary, and using *Data not available* map tiles for areas outside the city boundary. The major difference is that JPEG was used as the map tile image format, because it is recommended for imagery and generated map tiles that were four to five times smaller in size than PNG 32. Building the map cache for the Imagery Basemap took approximately 1 hour and 38 minutes.

[12] More information on selecting the appropriate image format for caches can be found in the ArcGIS Server Help topic Choosing cache properties.

[13] More information about automating cache updates can be found in the ArcGIS Server Help topic Automating cache creation and updates with geoprocessing.

Table 5 summarizes the map cache format choices made for the two map services previously discussed.

Table 5
Summary of Map Caches Built for the Riverside Viewer Web Application

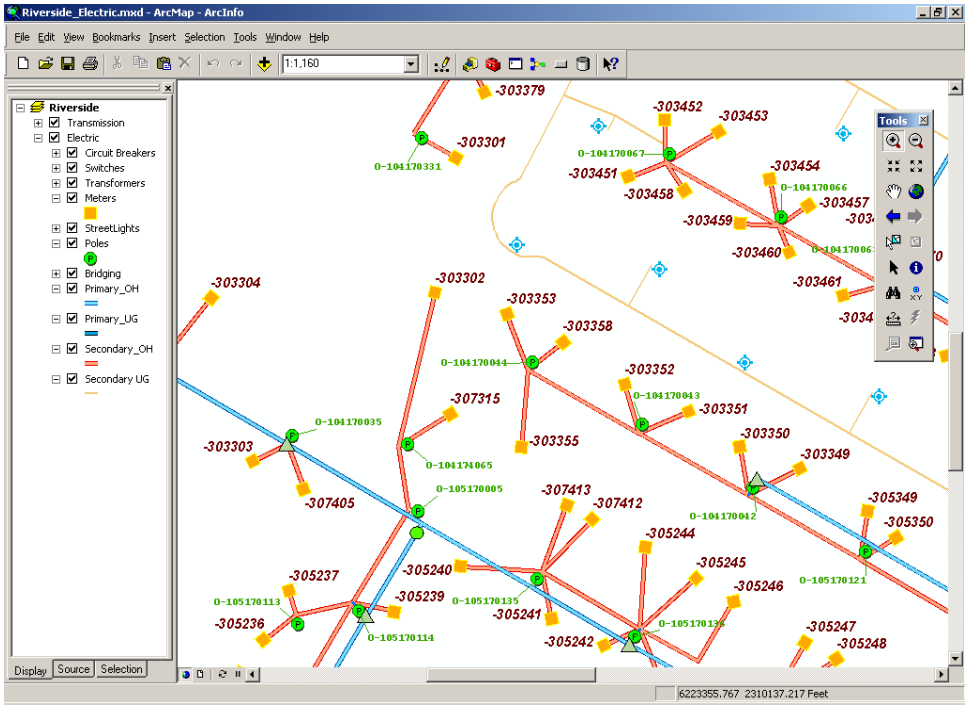
Web Service	Type of Service	Image Format	Tile Size	Explanation
Riverside Street Map	Map	PNG 8	~16 KB	Ideal for basemaps with few colors (exactly 256); yielded quality images with small tile size
Imagery Basemap	Map	JPEG	~200 KB	Ideal for raster-based maps with lots of color variation

Utilities Map Service

The Riverside_Electric map contains over 10 different layers such as poles, switches, bridges, transmission lines, meters, and so forth. One requirement of the Riverside Viewer Web mapping application is that users should be able to toggle and access content between these data layers as needed for their business requirements. Within a cached map service, data layers cannot be toggled, since all the map contents are aggregated (i.e., fused) into the map tiles. Another requirement is that specific assets in the map (e.g., poles and meters) must always be displayed in the map, along with text showing their unique identifier values. Because of these requirements, this map service is configured as a dynamic map service.

Another benefit of using a dynamic map service is that all the information is displayed/accessed live from the geodatabase. Changes to the source data are immediately visible when the map is next refreshed in the Web application. This is significant because utility network data is frequently updated. Figure 10 shows the Utilities dynamic map service as it appears in ArcMap.

Figure 10
Utilities Dynamic Map Service



Dynamic map services require careful planning, because they can stress resources on the SOCs and the geodatabase. Dynamic map services should typically only be used for operational layers in an ArcGIS Server Web application. As a general rule, dynamic map services should be used sparingly and only when business needs require their use. Keep their data content minimal. Applying scale dependencies is also strongly recommended for dynamic map services. These help control the number of features rendered at a given scale (e.g., only display features that make sense from a cartographic perspective), and therefore help accelerate rendering in the Web application.

Since the basemaps (Riverside Street Map and Imagery Basemap) are separate from the utility network information, the dynamic map service is quite simple. This is one of the most important decisions that impacts the performance and scalability of the Riverside Viewer Web application: by carefully separating the Web application content into cached and dynamic services, the effort needed to properly tune the Utilities dynamic map service is reduced.

Tip 5: If the same data can be contained in a cached map service or a dynamic map service, place it in a cached service—this is optimal. Avoid data duplication between cached map and dynamic map services.

It is strongly recommended that users tune dynamic map services as much as possible.[14] ArcGIS Server has the ability to render many maps concurrently, but the speed of rendering a map document is partly determined by how well it has been optimized for publication. If a map document has not been properly tuned before it is published, it will affect the performance and scalability of the entire ArcGIS Server system.

In this case, the Utilities map document was refined and adjusted to make it as optimal as possible until consistent subsecond map refresh response times were obtained. The most significant impact for optimizing a dynamic map service is to keep its contents simple (e.g., by offloading content to cached map services). Other aspects that can affect performance include building spatial indexes on the data, keeping geodatabase statistics up-to-date on the individual feature classes, using scale dependency, and minimizing dynamic labeling.

ArcGIS Desktop 9.3.1 introduces a new Map Service Publishing toolbar in ArcMap that helps properly tune map documents for publication in ArcGIS Server.[15] The Analyze option on this toolbar will loop through all the layers in a map document, identify any possible issues, and suggest tips to optimize the map's performance in ArcGIS Server. Once potential issues are resolved, the map document can be exported to a map service definition (.msd) file. ArcGIS Server dynamic map services created from .msd files are known as optimized map services.

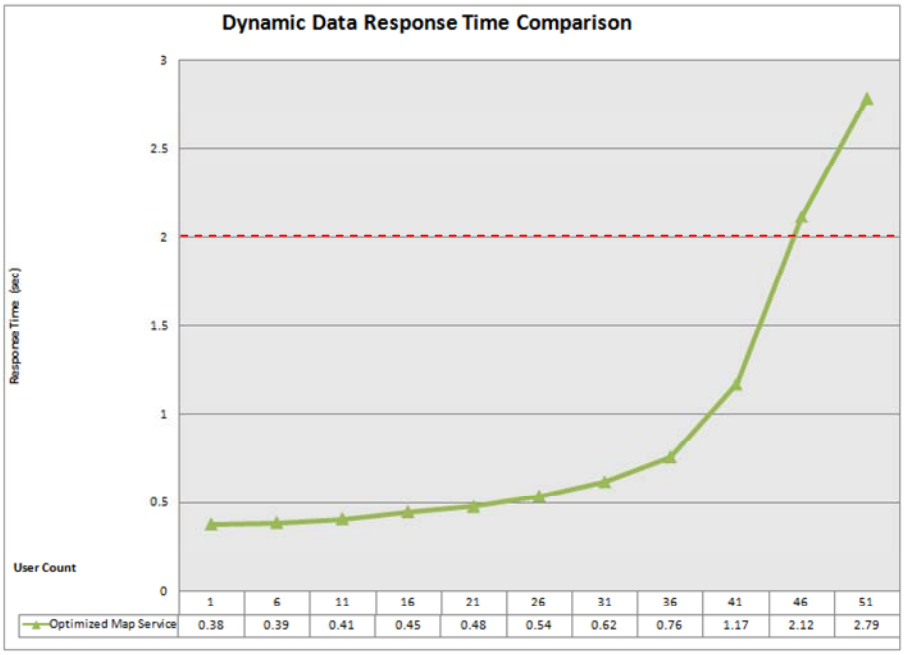
Tip 6: When possible, publish dynamic map services using map service definition (.msd) files. They are known as optimized map services and provide superior throughput.

Recall that earlier, in figure 5, a performance comparison chart between a dynamic map service based on an .mxd file and a dynamic map service based on an .msd file was shown. The optimized map service (based on the .msd file) consistently yielded faster response times as user count increased. Optimized map services perform better than .mxd-based dynamic map services. Therefore, the Utilities map document was implemented as an optimized map service based on an .msd file. A small test that simulated user load was applied to the service. Figure 11 shows its response time performance as the user count increases.

[14] More information on tuning dynamic map services can be found in the ArcGIS Server Help topic Map authoring considerations for ArcGIS Server.

[15] More information on the new Map Service Publishing toolbar can be found in the ArcGIS Desktop Help topic Publishing optimized map services.

Figure 11
Response Time of the Utilities Optimized Map Service



Again, a threshold response time of two seconds was set as the maximum acceptable response time for the service. The Utilities optimized map service has very good throughput. It yields response times consistently below half a second for up to 21 users and below one second for up to 36 users and remains under the acceptable two-second threshold until reaching approximately 46 users.

At this point, all three map services are now defined to support the requirements of the scenario. All three can be combined in different ways in the final Riverside Viewer Web mapping application. It is important to highlight that these map services serve a dual function: they are the building blocks of the maps displayed in the Web application and help provide the Web application's query capabilities. With map queries, additional information on the data participating in the map services can be found, such as locating a specific parcel, street, or meter.

U.S. Street Geocoding Service

The fourth Web service used in the Riverside Viewer Web mapping application is a geocoding service. This service satisfies the user workflow of being able to find an address in the city. A U.S. street address style (i.e., a house number followed by the street name) is used to define the address structure for the address locator.[16] Two feature classes in the available city of Riverside GIS dataset could potentially be used as reference data for the locator: Addresses, a point feature class with all the city addresses, and Streets, a line feature class with address information. The feature classes are shown in figures 12 and 13, respectively.

[16] More information on address locators can be found in the ArcGIS Desktop Help topic Definition of the address locator.

Figure 12
Sample Addresses of the Addresses Point Feature Class

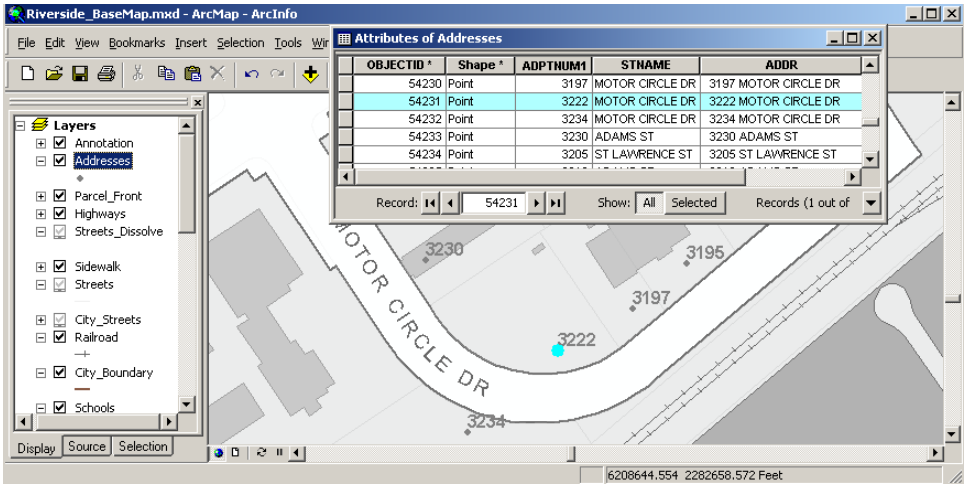
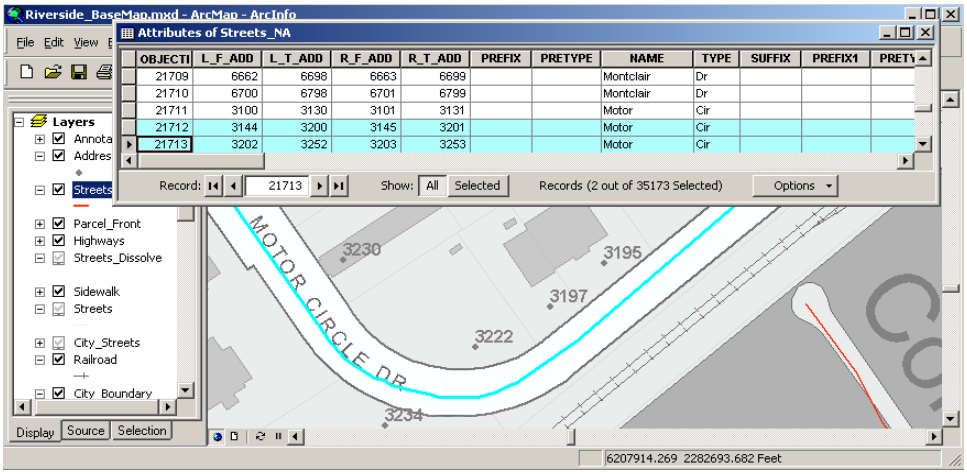


Figure 13
Sample Addresses of the Streets Line Feature Class



To determine which of the two feature classes should be used as the reference dataset for the address locator in the geocoding service, two separate locators were built (one for each feature class) and compared in terms of their relative performance in ArcGIS Server.

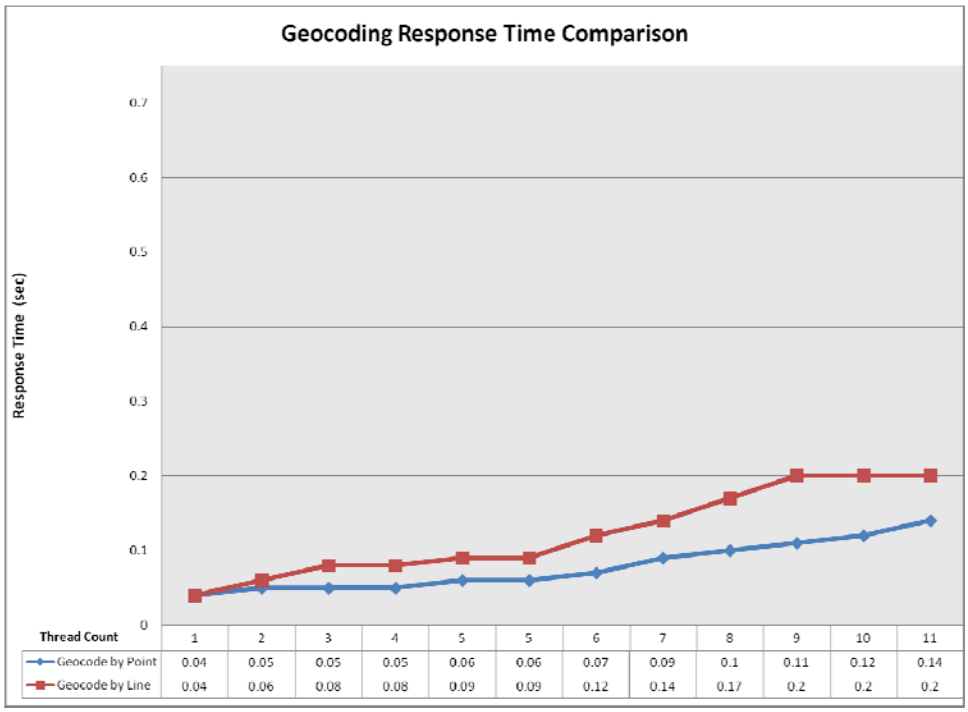
In this case, the performance of the two geocoding services was assessed in terms of *thread*[17] *count*. This means performance was measured in terms of raw throughput numbers and not in terms of a simulated user load on the services. This is a different type of performance measurement and is designed to gauge performance differences between services. It is not a user capacity measure of the services. Therefore, the results in

[17] In this document, a thread is defined as a consumer of data services within an Internet application that requires no pause (i.e., think time) between interactions to decide if further interaction is needed.

figure 14 are not directly comparable with the results shown earlier in figures 5 and 11, which assessed performance in terms of number of users (i.e., capacity assessment).

The maximum number of locator service instances was set to two, to minimize the consumption of server resources while ensuring acceptable throughput. Figure 14 illustrates the geocoding throughput response time results of the two address locators.

Figure 14
Comparison of Geocoding Throughput Results between the Point and Line Locators



Thread Count	1	2	3	4	5	5	6	7	8	9	10	11
Geocode by Point	0.04	0.05	0.05	0.05	0.06	0.06	0.07	0.09	0.1	0.11	0.12	0.14
Geocode by Line	0.04	0.06	0.08	0.08	0.09	0.09	0.12	0.14	0.17	0.2	0.2	0.2

Observe that both address locators generate response times of under 0.2 seconds, which is great performance. The point feature class-based locator yielded better geocoding throughput results overall when compared with the line feature class-based locator in terms of average response time as the thread count increased. The point-based locator yielded a peak throughput of approximately 279,944 geocoded addresses per hour, while the line-based locator yielded a peak throughput of approximately 230,313 geocoded addresses per hour. This suggests that the point feature class-based locator would likely, on average, have less impact on the overall ArcGIS Server system.

Geometry: Map and Geometry Service

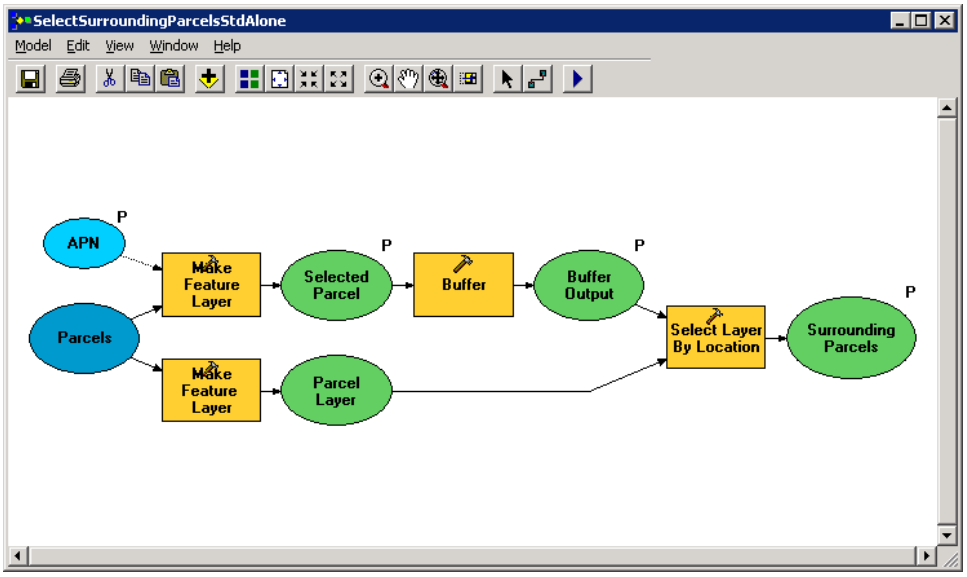
The geometry service helps support the reporting tool in the Riverside Viewer Web mapping application. It enables users to enter the Assessor Property Number (APN) of a city parcel and retrieve a list of all parcels within 300 feet of the selected parcel. This is a very common task performed in municipal/local government. An example use case for this operational task is the creation of a notification list after a land-use change is proposed for a property.

There are two possible approaches to incorporate this functionality into the Web application: (1) using a geoprocessing service or (2) with a combination of map queries and a geometry service (available since the 9.3 release). Both options are discussed and their relative performance compared.

For the first approach, a geoprocessing service enables geoprocessing models authored with ArcGIS Desktop to be published as Web resources that can be used by Web applications. Geoprocessing in ArcGIS is a very powerful framework for developing sophisticated workflows for geographic analysis.[18] Tasks are created by publishing geoprocessing toolboxes or map documents containing tool layers. They execute on the server using resources on the server computer.[19]

In this case, a model named SelectSurroundingParcelsStdAlone was created in ModelBuilder™ (see figure 15), then published as a stand-alone toolbox. Its execution mode was set to synchronous, because the job sent to the server is short in duration. The model takes an input APN value, extracts the requested parcel from the Parcels data layer, selects the surrounding parcels that are within 300 feet around it with a buffer, and finally exposes them as a FeatureRecordSetLayer parameter. It also renders the result set in the map display. Appendix D contains a detailed explanation of the SelectSurroundingParcelsStdAlone model and how it works.

Figure 15
SelectSurroundingParcelsStdAlone Geoprocessing Model



[18] More information on geoprocessing in ArcGIS can be found in the ArcGIS Desktop Help topic What is geoprocessing?

[19] More information on geoprocessing services can be found in the ArcGIS Server Help topic An overview of geoprocessing with ArcGIS Server.

The alternative approach consists of using the query capabilities of the Riverside Street Map map service to query the parcels layer along with a geometry service. In this type of workflow, the Riverside Viewer Web mapping application is responsible for cascading the client requests:

1. Search a parcel by its APN and return its geometry using the query capabilities of the Riverside Street Map map service.

2. Buffer the selected parcel to a distance of 300 feet with the geometry service.

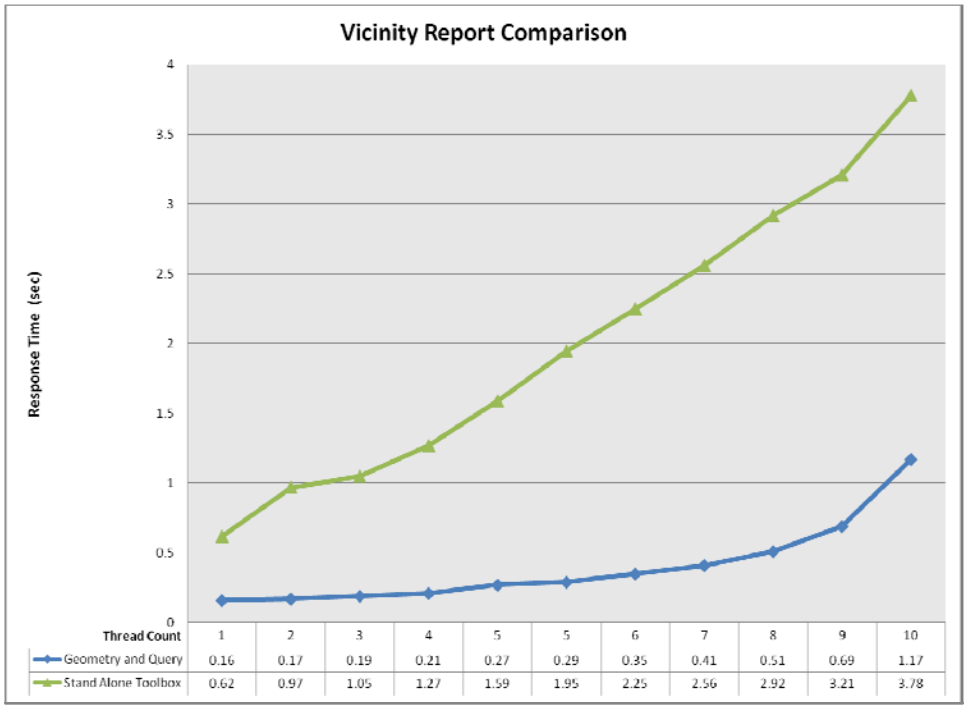
3. Use the buffered geometry to spatially query the parcels layer again and return all parcels that intersect the buffered area.

4. Render the result set in the map display.

Appendix E contains more detailed information on implementing this approach.

Both approaches to enabling the reporting task functionality in the Riverside Viewer Web mapping application were implemented and compared. To determine which was better, performance between the two approaches was measured in terms of throughput. Similar to the comparison test between the two geocoding services in figure 14, the two reporting task approaches were measured based on thread count (see figure 16).

Figure 16
Comparison of Geoprocessing Service Response
vs. Map + Geometry Service Response



Thread Count	1	2	3	4	5	5	6	7	8	9	10
Geometry and Query	0.16	0.17	0.19	0.21	0.27	0.29	0.35	0.41	0.51	0.69	1.17
Stand Alone Toolbox	0.62	0.97	1.05	1.27	1.59	1.95	2.25	2.56	2.92	3.21	3.78

The performance difference between the two approaches was significant. The map query and geometry service yielded much faster response times over the geoprocessing service throughout the test. As the thread count increased, the response time for the map query and geometry service was consistently under half a second until eight or more threads were applied. The geoprocessing service response time was consistently more than half a second and continued to increase as more threads were added. The geoprocessing service tends to be resource intensive in nature and causes the ArcGIS Server system to utilize more resources to execute the task. This can adversely affect other incoming requests on the server.

It was observed that the maximum throughput for the geoprocessing service was approximately 10,896 transactions per hour versus 98,064 transactions per hour for the map query and geometry service approach. Therefore, the map query and geometry service approach was selected for the Riverside Viewer Web mapping application, because of its reduced system-level impact and simplicity of use.

Tip 7: Use geometry services for simple operations such as buffering and spatial selections. Use geoprocessing services for more sophisticated operations.

Web Application Development

ArcGIS Server provides several different software developer kits (SDKs) for building Web applications. There are two groups of SDKs:

- ArcGIS Web Mapping APIs
  - JavaScript™ (www.esri.com/javascript)
  - Flex™ (www.esri.com/flex)
  - Silverlight™ (www.esri.com/silverlight)
- Web Application Development Frameworks (ADF™)
  - .NET (www.esri.com/net)
  - Java™ (www.esri.com/java)

The ArcGIS Web Mapping APIs provide a simple programming model for building applications. Their simple architecture and approachable SDK allow Web developers to quickly and efficiently build Web mapping applications. Web ADFs provide a more sophisticated development environment for GIS-based Web applications.

Tip 8: The most important factor when deciding which SDK to use for ArcGIS Server development is skills: users should select the SDK based on how comfortable they are with one programming environment as opposed to another. Alternatively, if they have no programming skills, they can configure the out-of-the-box Web mapping application.

As mentioned previously, the ArcGIS API for Flex was used to create the Riverside Viewer Web mapping application. The application is based on a sample viewer that can be found online at the ArcGIS Server Resource Center.[20]

[20] More sample viewers for the ArcGIS API for Flex are available at http://resources.esri.com/arcgisserver/apis/flex/.

In terms of mapping, to simplify the user experience, two exclusive basemaps (Riverside Street Map and Imagery Basemap) were configured. By design, users are purposely prevented from being able to display both basemaps simultaneously. This prevents the client from downloading map tiles for both services when navigating the map. While it is possible to enable the two map services to work in combination, this would have required both cartographic and performance trade-offs.

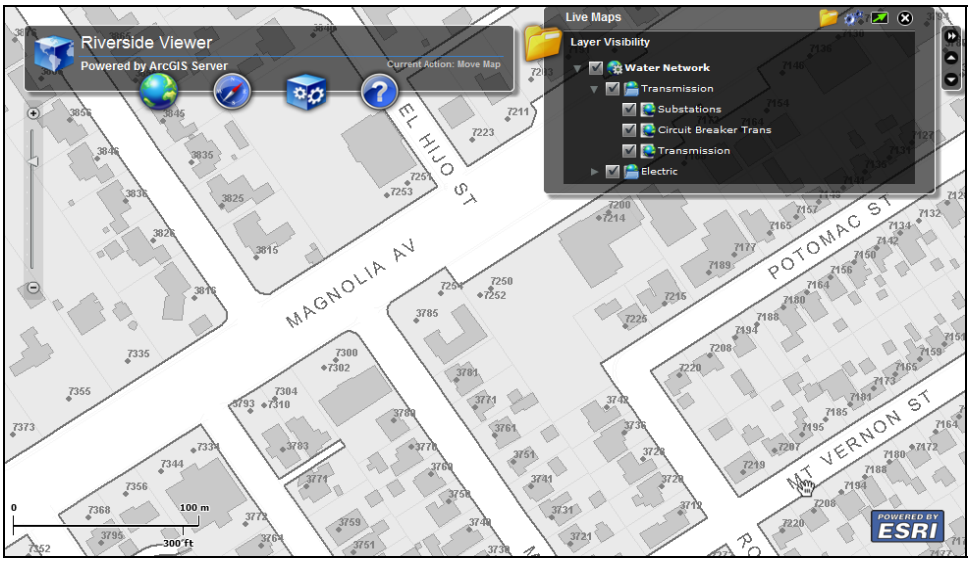
The Utilities dynamic map service will always display on top of the basemaps and offers users the ability to turn its layers on and off in the display. Layers from the map document were logically grouped into operational layers, thus supporting the visibility of themes as opposed to layers. Table 6 lists the data layers in the Riverside_Electric map document and how they are aggregated into operational layers for the Web application.

Table 6
List of ArcMap Layers That Can Be Toggled in the Riverside Viewer Web Mapping Application

ArcMap Layers	Web Application Layers That Can Be Toggled
Circuit Breaker Transmission	Transmission
Substations	
Transmission	
Bridging	Electric
Circuit Breakers	
Streetlights	
Switches	
Transformers	
Poles	Poles
Meters	Meter
Primary OH	Distribution
Secondary OH	
Secondary UG	

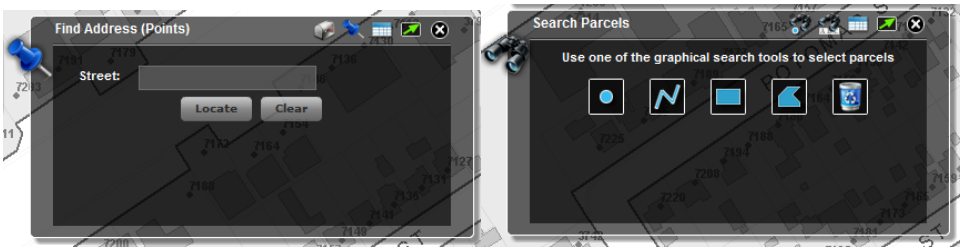
The user experience for the Riverside Viewer Web mapping application is simple and intuitive. Controls are located in the upper left corner, and drop-down menus appear when the cursor is paused on one of the four control icons: Map, Navigation, Tools, and Help. Additional control windows may appear in the upper right corner of the application when activated. Appendix A contains information on the functionality available in the Web mapping application and numerous screen captures that illustrate the example user workflow described in table 1. Figure 17 shows the selectable data layers in the Web application from the Utilities dynamic map service.

Figure 17
Riverside Viewer Web Mapping Application
Showing Selectable Layers in the Utilities Service



Additional capabilities of the Riverside Viewer Web mapping application include Find Address, Search Parcels, Search Schools, and Search Pole by ID. Some of these search windows in the Web application are shown in figure 18.

Figure 18
Riverside Viewer Web Mapping Application Example Searches



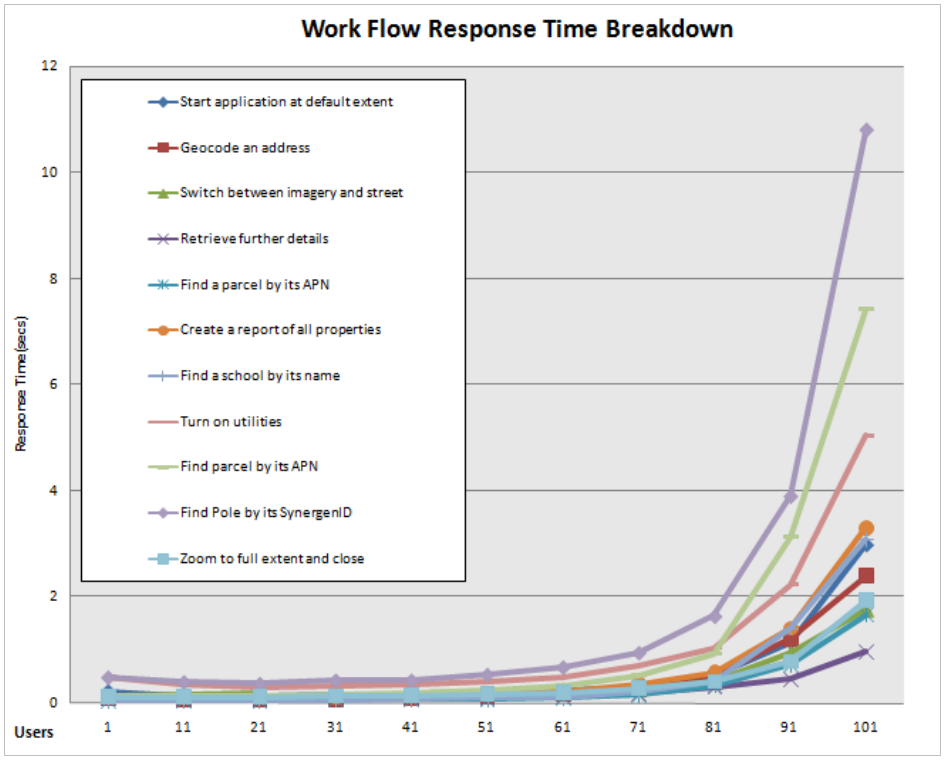
ArcGIS Server Solution Assessment

This section discusses how the overall ArcGIS Server system works when all the pieces are finally assembled—in other words, how the system behaves when many simultaneous users access the Riverside Viewer Web mapping application to perform their business task requirements. Microsoft® Visual Studio® Team Systems 2008 Test Edition was the software used to record and simulate the Web traffic between the server and client machines. Appendix F contains details on the performance testing methodology.

The example user workflow outlined in table 1 was performed on the server. Multiple users were applied to simulate heavy load on the server, and the response time for each individual step in the example workflow was measured. A think time of six seconds was set between each workflow step, along with random input APNs and addresses to further simulate a more realistic user load scenario. In effect, capacity performance and system

scalability of the ArcGIS Server system was measured. Response time performance results are shown in figure 19.

Figure 19
Performance Chart of Different User Workflow Tasks



The ArcGIS Server system was able to support all tasks from the example user workflow with response times that were less than two seconds—the acceptable threshold response time. This response time result was consistent until approximately 81 or more users were accessing the Web application. Performance measurement of the configuration indicated that at 81 users, the system's CPU usage was at approximately 80 percent. This is a good indicator that the ArcGIS Server solution was designed well and optimally configured for a peak of no more than 81 concurrent users.

The three most expensive tasks (in terms of highest response time) from the example user workflow were (in decreasing order): find pole by its SynergenID (step 12), find parcel by its APN (step 10), and turn on utilities data (step 8). These tasks are directly associated with the Utilities dynamic map service, which was expected to require more resources. Recall that in dynamic map services, the information content is displayed/accessed live from the geodatabase. Performance for these steps can be easily improved by adding more SOC machines to the configuration.

Overall, the completed Riverside Viewer Web mapping application is a successful ArcGIS Server solution for the example municipal/local government use case. With good planning and efficient Web services design, the resultant Web mapping application satisfied all the business requirements of the use case.

Summary

To conclude, this document discussed several best practices for an ArcGIS Server Web mapping application design for municipal/local government use. It also described the logic behind certain implementation/configuration decisions. Below is a summary of the high-level Web mapping application design tips that were presented and discussed:

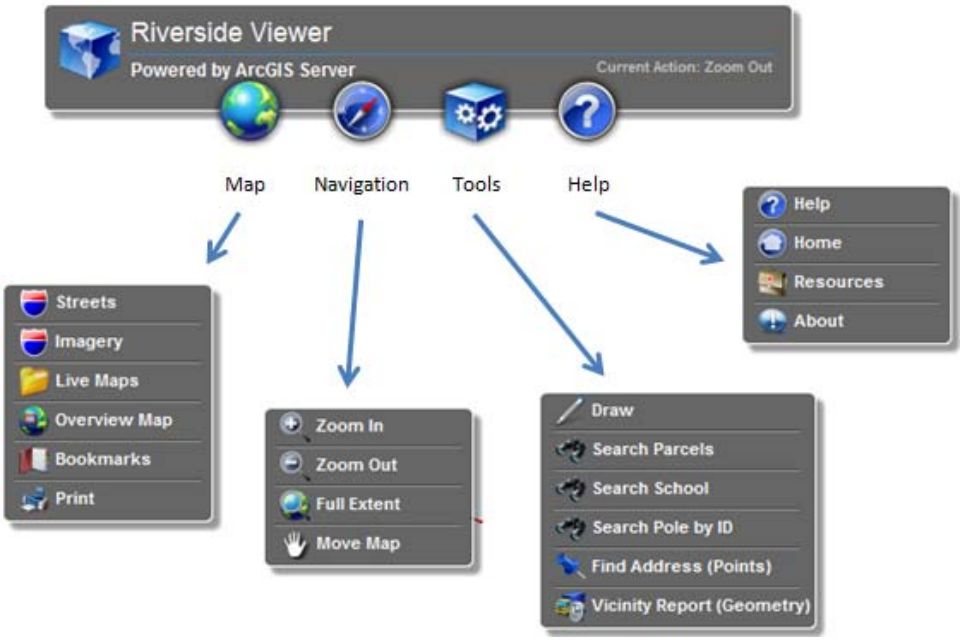
- Use cached map services for serving GIS data with ArcGIS Server whenever possible. They yield better performance and put less load on the server compared to dynamic map services.
- In general, avoid caching areas in the map that will not be used. This will not only save disk space, but more importantly, it will help expedite the map cache building process (i.e., make it much faster).
- Esri recommends being conservative when building partial caches. Users should benefit performance-wise from the map cache in most instances, and trigger on-demand caching rarely or never. When visiting areas that have not been cached, users will suddenly notice that the map navigation becomes slightly sluggish, as the map tiles are created on the fly.
- Imagery map caches should be separate from vector data map caches. Do not combine imagery and text data within the same cache; instead, create one cache for imagery and a separate cache for text (e.g., annotation).
- If the same data can be contained in a cached map service or a dynamic map service, place it in a cached service—this is optimal. Avoid data duplication between cached map and dynamic map services.
- When possible, publish dynamic map services using map service definition (.msd) files. They are known as optimized map services and provide superior throughput.
- Use geometry services for simple operations such as buffering and spatial selections. Use geoprocessing services for more sophisticated operations.
- The most important factor when deciding which SDK to use for ArcGIS Server development is skills: users should select the SDK based on how comfortable they are with one programming environment as opposed to another. Alternatively, if they have no programming skills, they can configure the out-of-the-box Web mapping application.

Appendix A: Riverside Viewer Web Mapping Application and User Workflow

The Riverside Viewer Web mapping application was created with the ArcGIS API for Flex, which can be easily configured to work with ArcGIS Server services. It is a free download from the ArcGIS Server Code Gallery in the ArcGIS Server Resource Center:

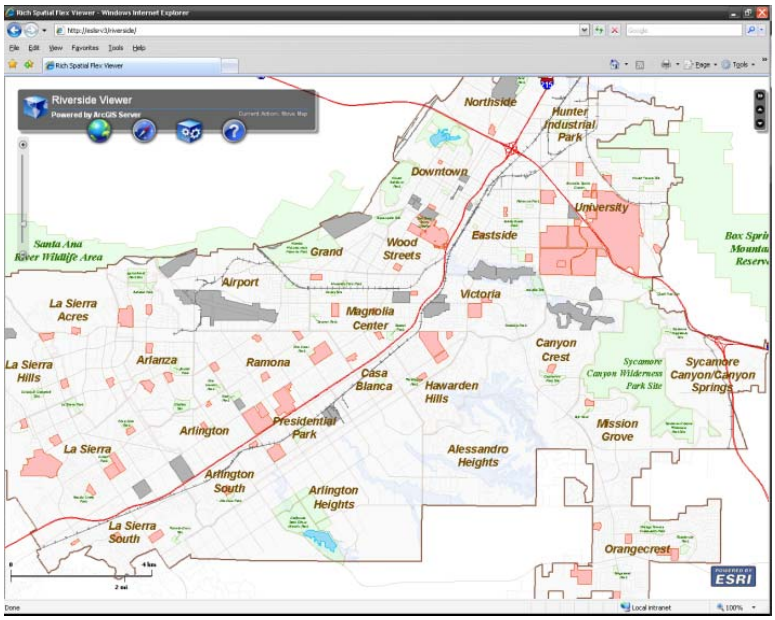
ArcGIS API for Flex version 1.1
http://resources.esri.com/arcgisserver/apis/flex/

Available Functionality in the Riverside Viewer Web Application

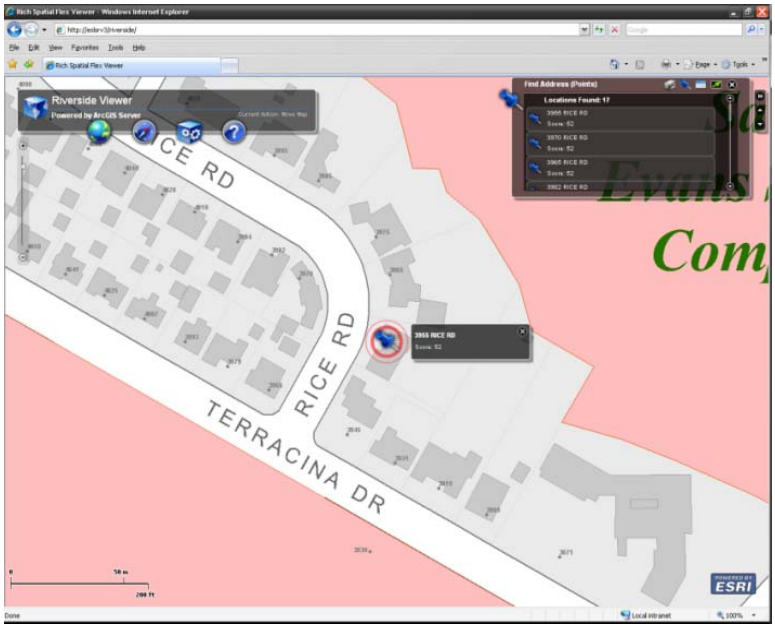


Screen Captures of the Example User Workflow

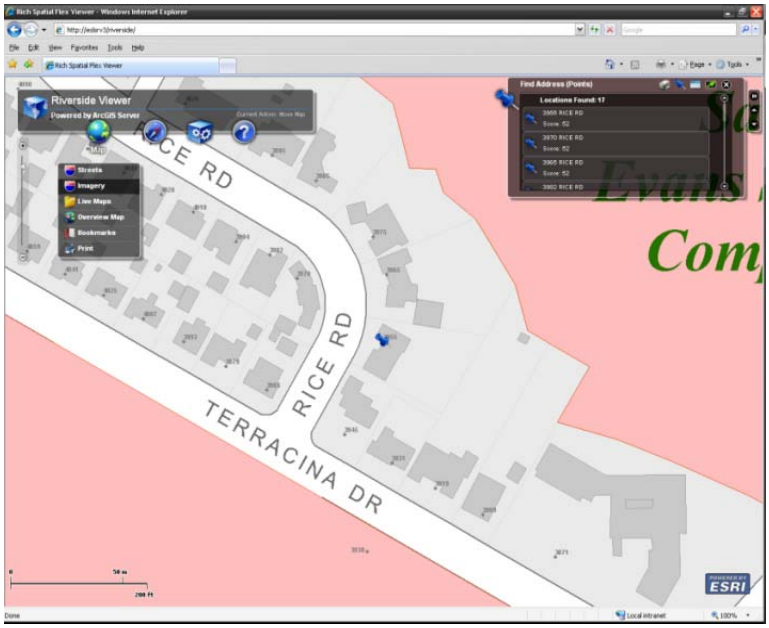
1. Start the application at the default extent: overview of the city of Riverside with the street map displayed.

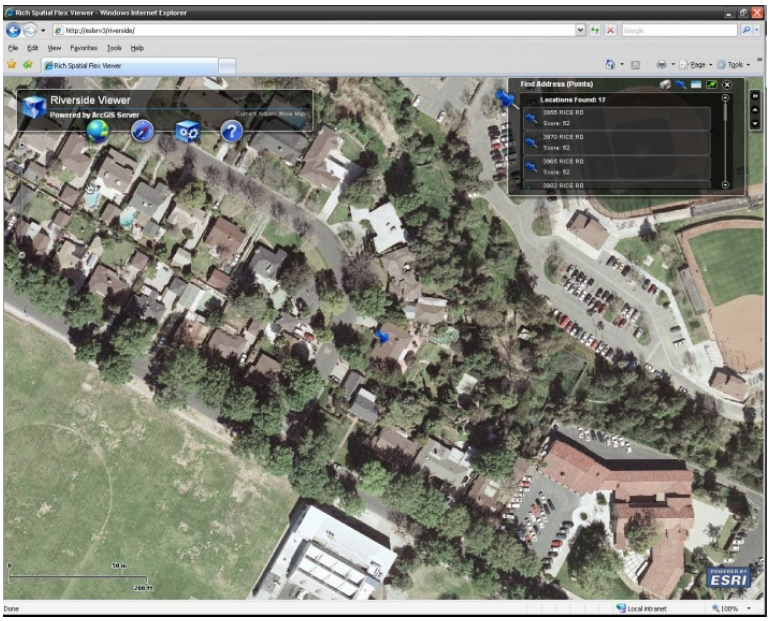


2. Geocode an address (e.g., 3090 Rice Road); center the map on the candidate result.

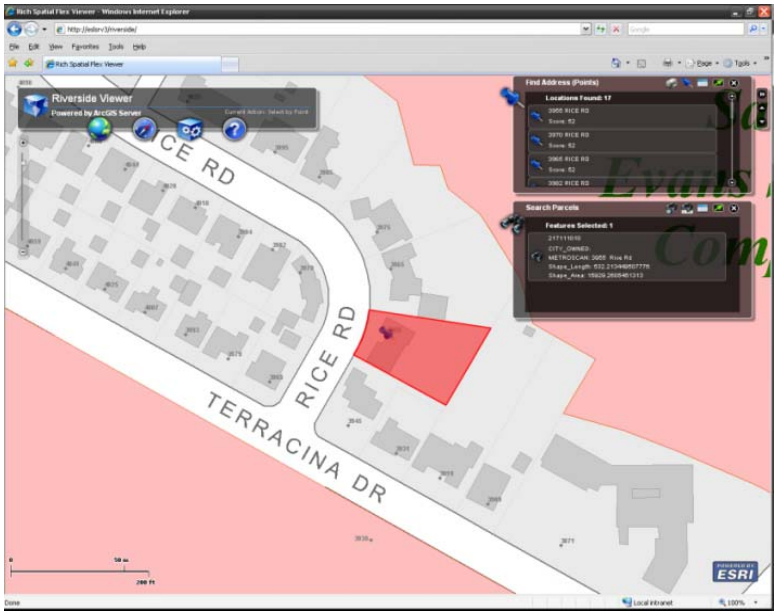


3. Switch the basemap to imagery, then switch back to street map.

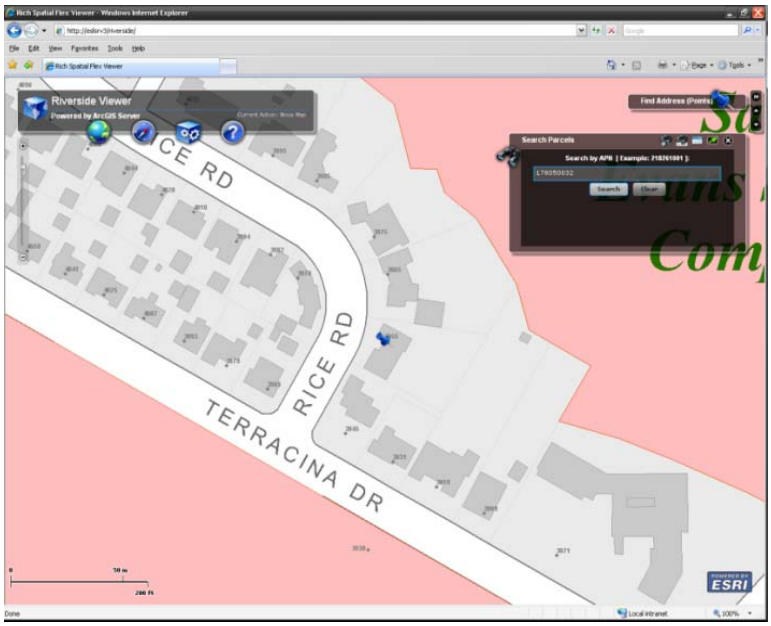




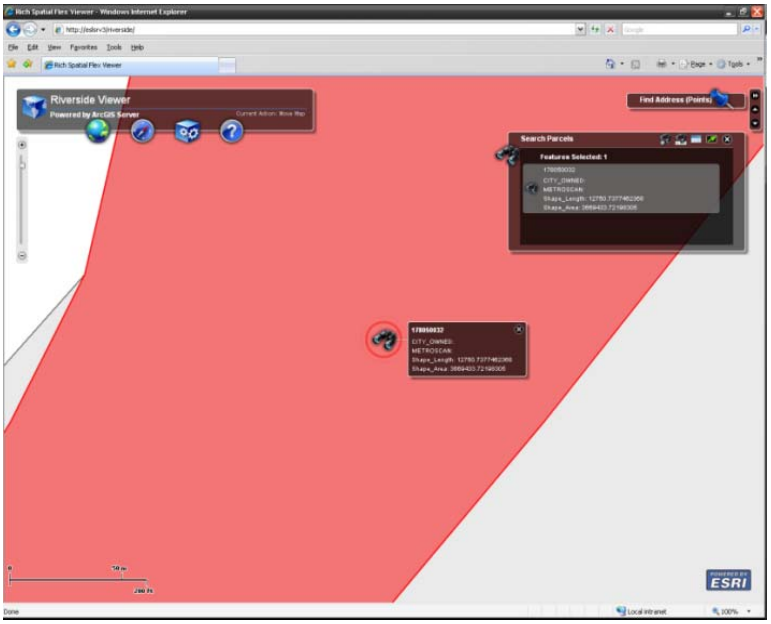
4. Retrieve further details on the parcel at the address by identifying it.



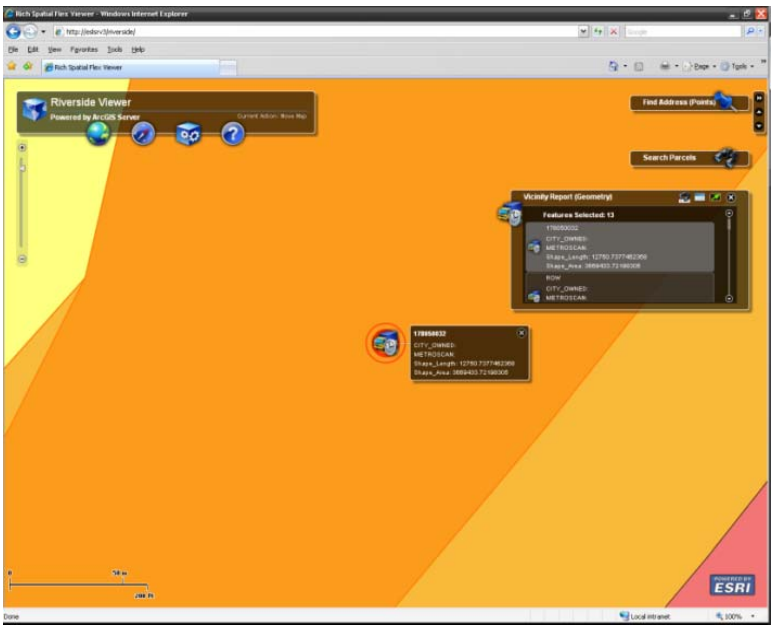
5. Find a parcel by its APN (e.g., 218181018) and zoom to it.



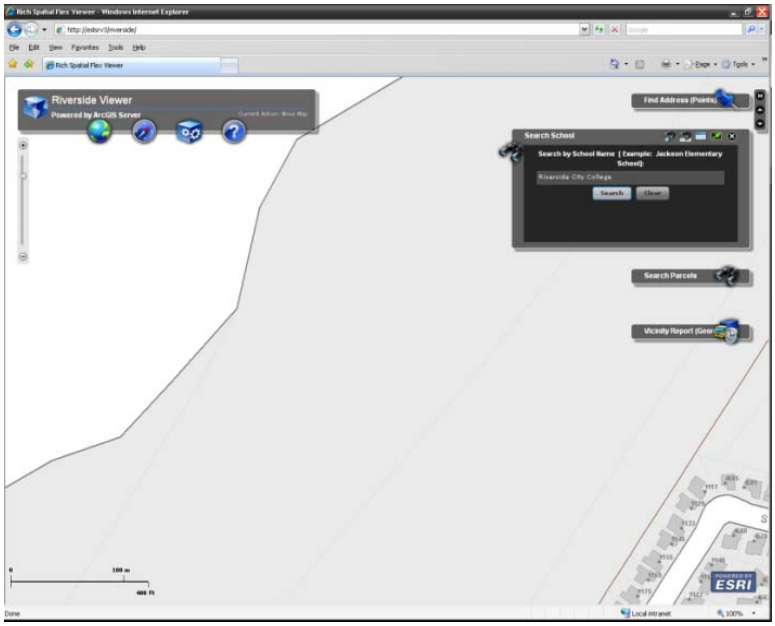
J-9804

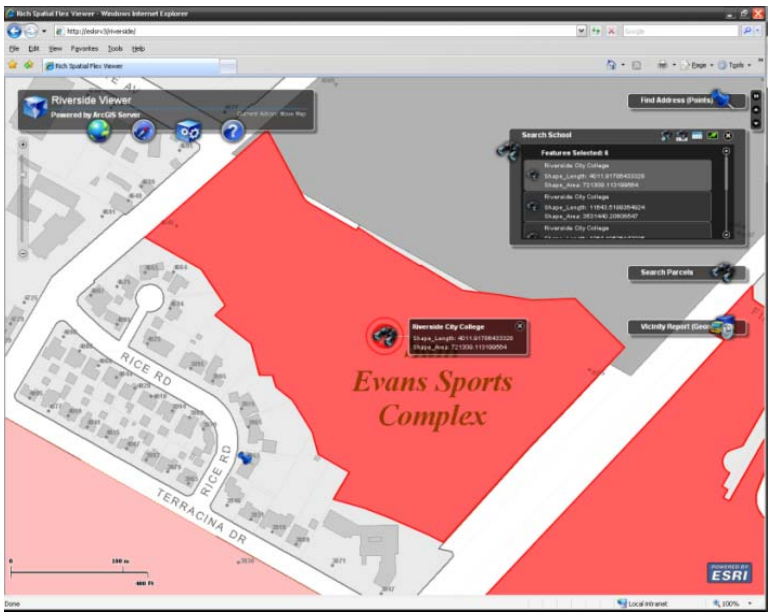


6. Create a report of all properties within 300 feet of the parcel.

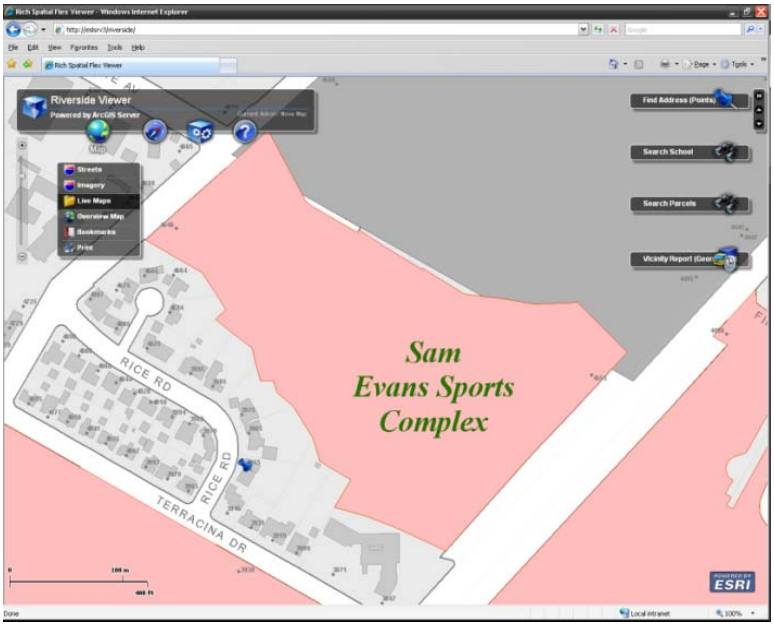


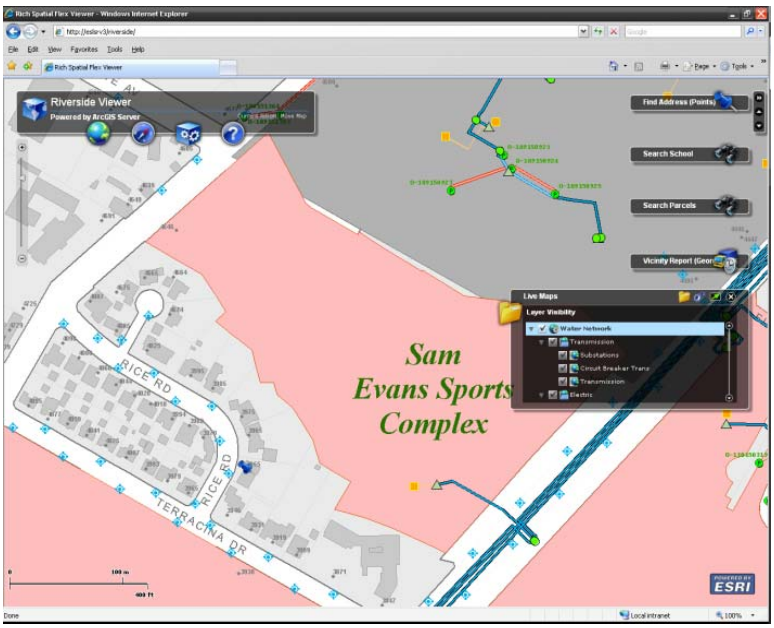
7. Find a school by its name (e.g., Riverside City College) and zoom to it.



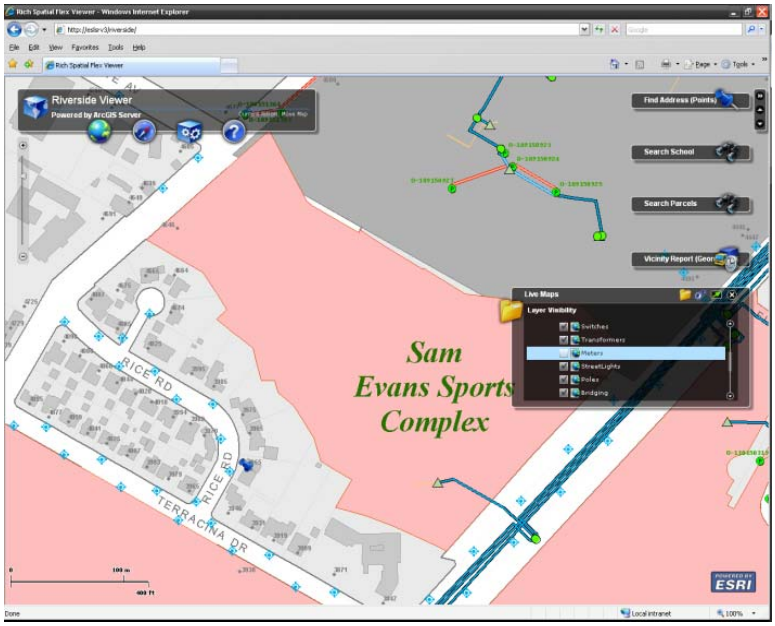


8. Turn on the Utilities dynamic layer.



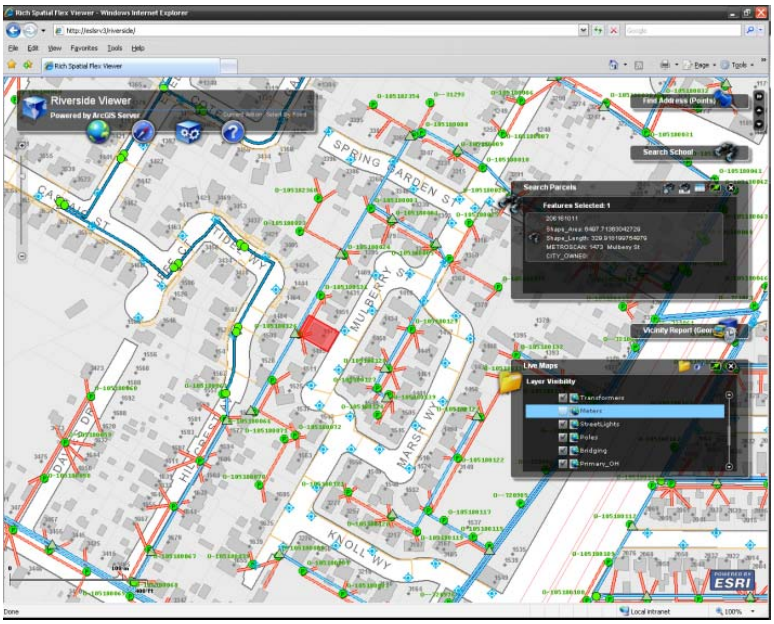


9. Turn off the meters layer and refresh the map display.

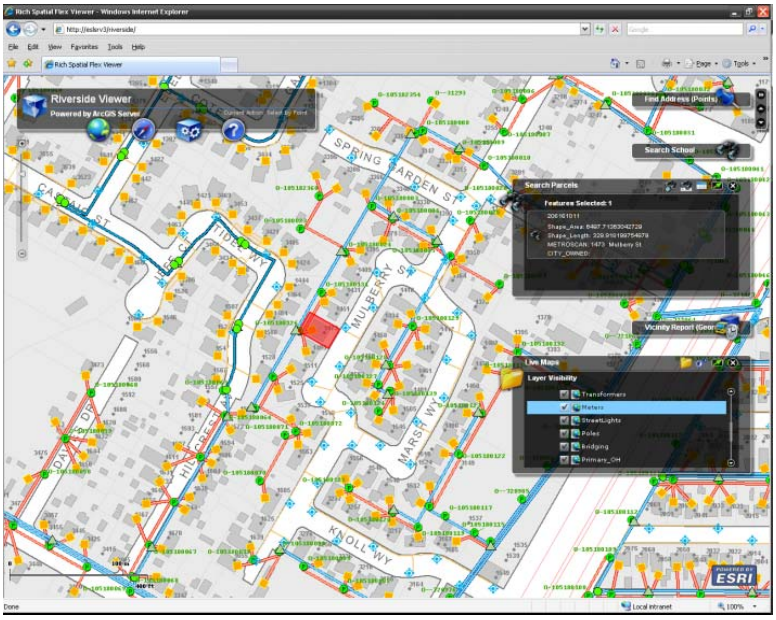


10. Find a parcel by its APN and zoom to it.

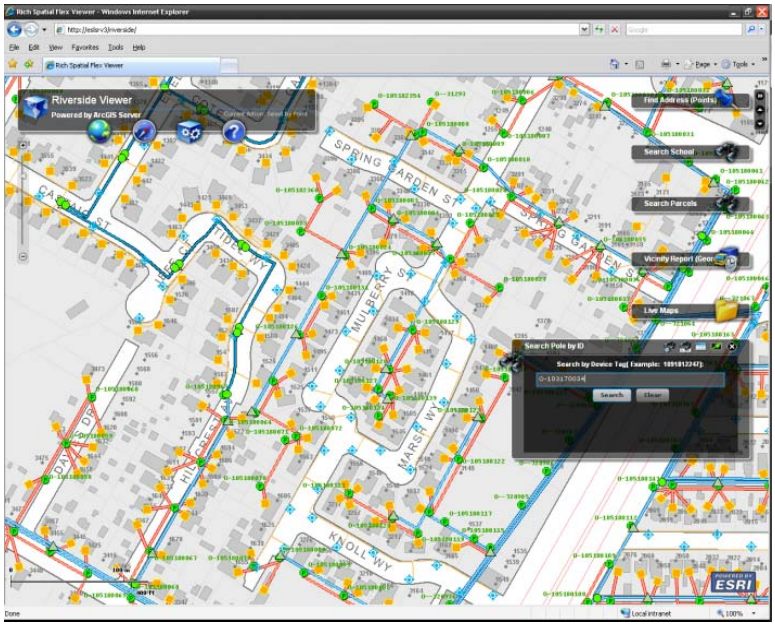
J-9804

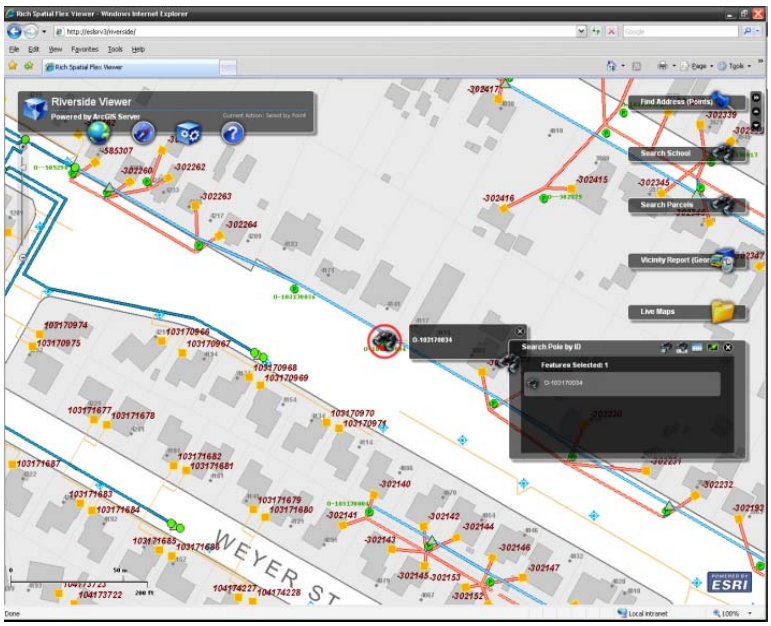


11. Turn on the meters layer.

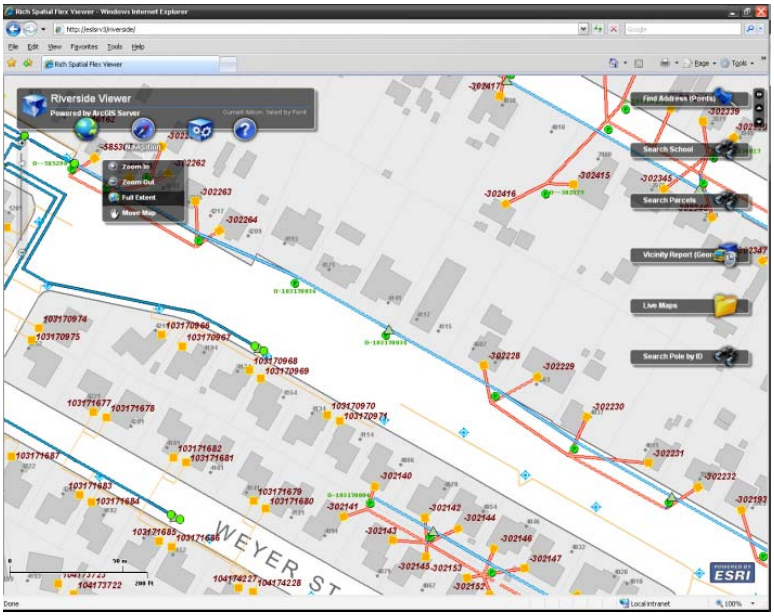


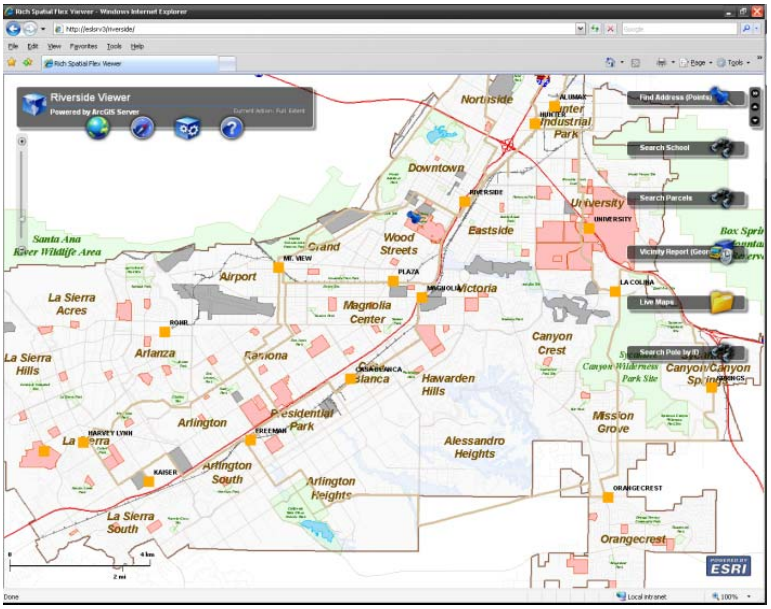
12. Find an electrical pole by its SynergenID and zoom to it.





13. Zoom to the full extent and close the application.





Appendix B: GIS Datasets for the City of Riverside, California

The Riverside Viewer Web mapping application used the following GIS datasets:

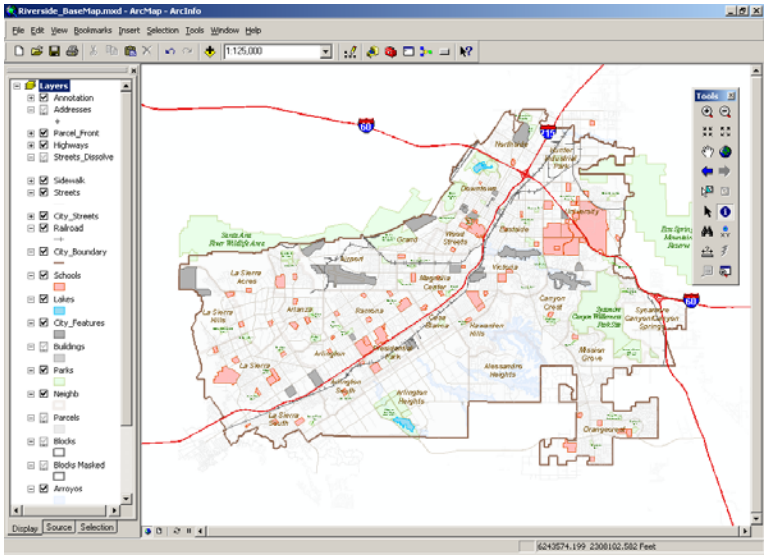
Dataset Name	Dataset Type	Size (# of records)
File Geodatabase		
Addresses	Point	84,213
Arroyos	Polygon	49
Blocks	Polygon	2,516
Buildings	Polygon	105,562
City_Boundary	Line	21
City_Features	Polygon	40
City_Features_Anno_10K	Annotation	24
City_Features_Anno_25K	Annotation	27
City_Features_Anno_50K	Annotation	32
City_Streets	Line	528
Highway_Shields	Point	5
Highways	Line	24
Neighb	Polygon	28
Neighb_Anno	Annotation	27
Parcel_Front_Anno_500	Annotation	69,432
Parcel_Front_Anno_500_Mask	Annotation	69,431
Parcels	Polygon	79,675
Parks	Polygon	78
Parks_Anno	Annotation	3
Parks_Anno_30K	Annotation	50
Parks_Anno_7500	Annotation	59
Railroad	Line	242
Schools	Polygon	87
Schools_Anno_15K	Annotation	69
Schools_Anno_25K	Annotation	69
Sidewalk	Line	21,831
Streets	Line	23,302
Streets_Anno_10K	Annotation	2,124
Streets_Anno_15K	Annotation	1,071
Streets_Anno_2500	Annotation	3,936
Streets_Anno_7500	Annotation	2,890
Streets_Dissolve	Line	4,077

J-9804

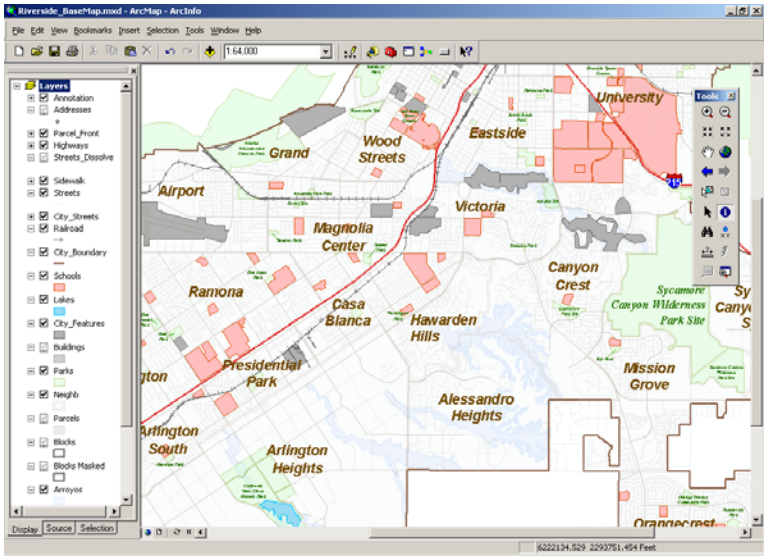
Dataset Name	Dataset Type	Size (# of records)
Workgroup ArcSDE Geodatabase		
Bridging	Line	2,474
Circuit_bkr	Point	38
Meters	Point	7,972
Poles	Point	4,559
Primary_OH	Line	2,505
Primary_UG	Line	3,034
Secondary_OH	Line	8,879
Secondary_UG	Line	6,828
Streetlights	Point	5,248
Substation	Point	16
Switch	Point	2,085
Trans_ckt	Point	4
Trans_line	Line	317
Transformers	Point	2,446

Appendix C: Riverside Street Map—Scale Cache Levels

Street Map at 1:125,000 Scale

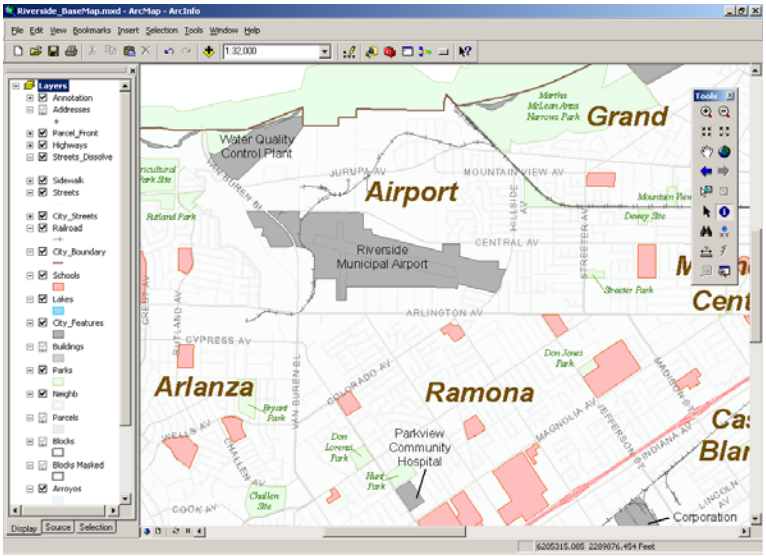


Street Map at 1:64,000 Scale

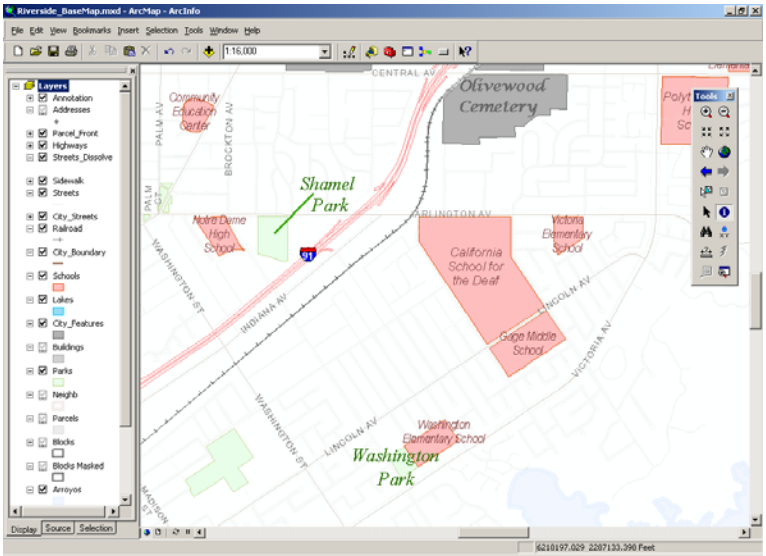


J-9804

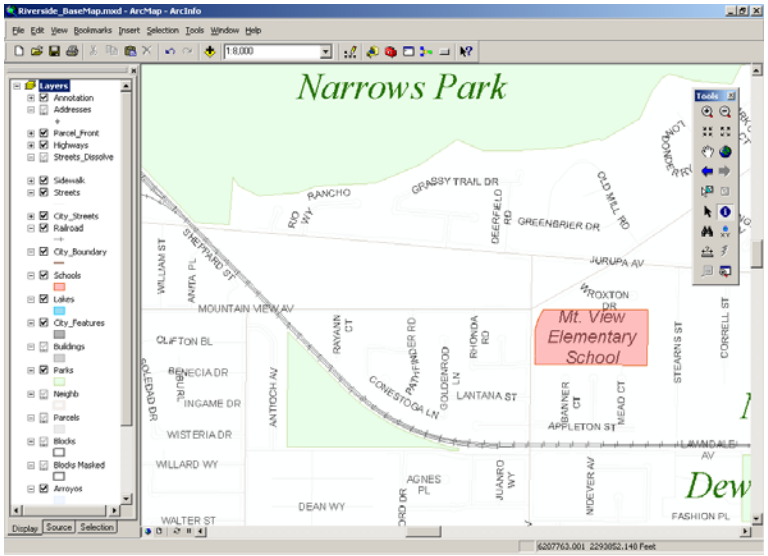
Street Map at 1:32,000 Scale



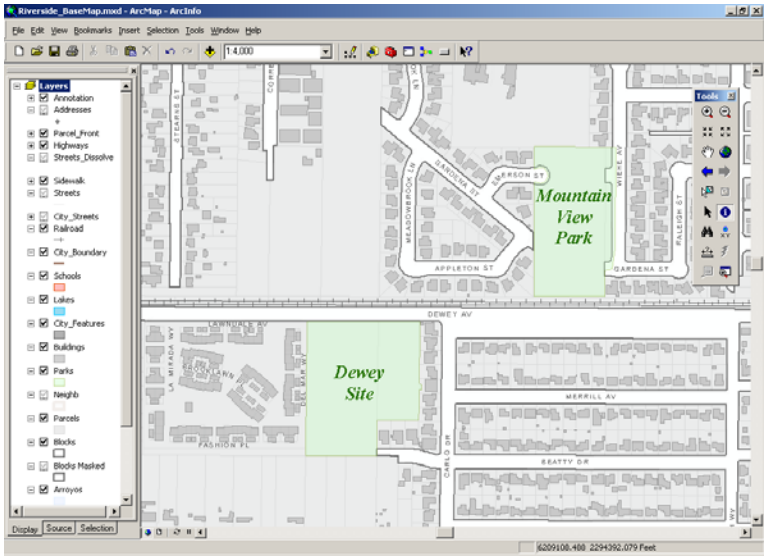
Street Map at 1:16,000 Scale



Street Map at 1:8,000 Scale

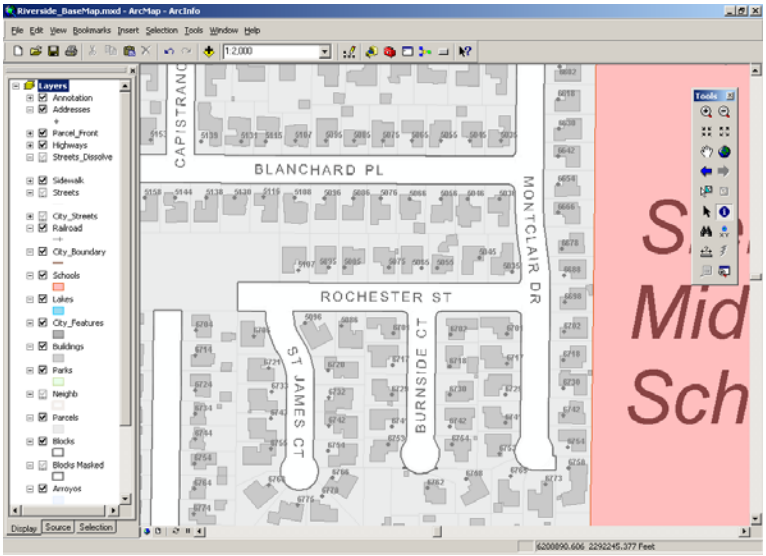


Street Map at 1:4,000 Scale

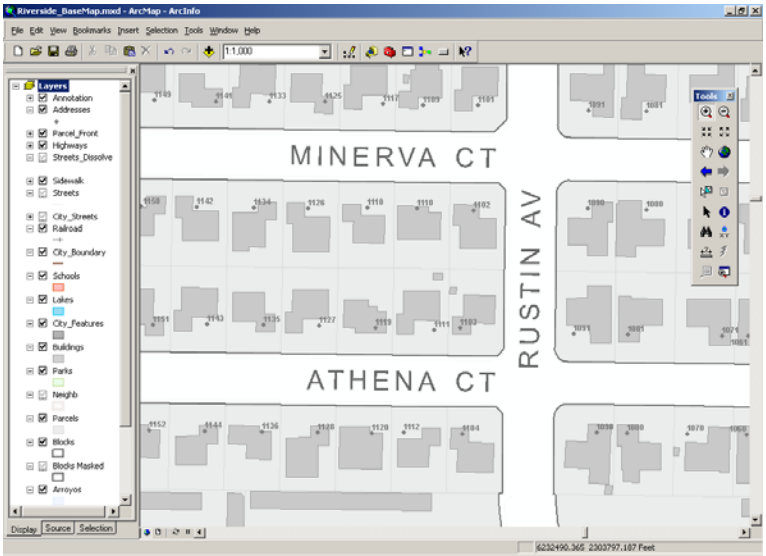


J-9804

Street Map at 1:2,000 Scale

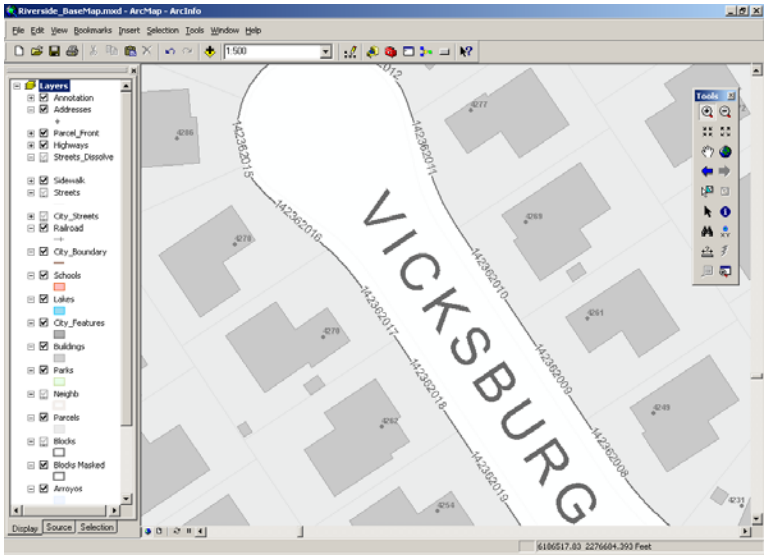


Street Map at 1:1,000 Scale



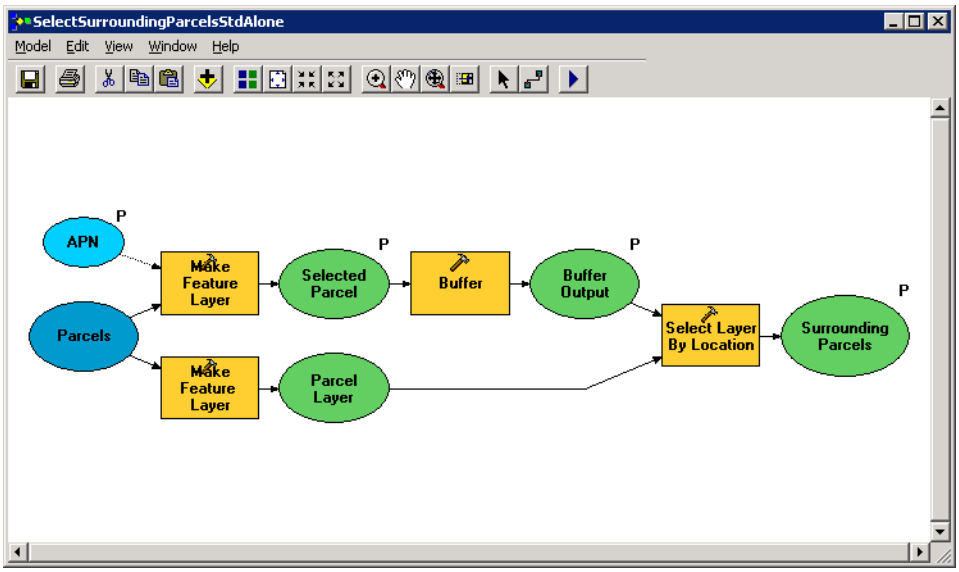
J-9804

Street Map at 1:500 Scale



Appendix D: The Geoprocessing Service

In this appendix, the workflow and logical order of operations in the SelectSurroundingParcelsStdAlone geoprocessing model are explained.



The model can be subdivided into three general steps:

- **Step 1**—An input APN value (to select a parcel) is sent to a Make Feature Layer tool. The expression in Make Feature Layer is `"APN" = '%APN%'`, which takes the input APN string parameter from the user and queries the Parcels data layer. To keep the model simple, only the APN string is exposed as a parameter to the user (instead of the entire SQL expression parameter). In general, exposing many SQL expression parameters in geoprocessing services will likely make them less intuitive to use and more error prone. The Make Feature Layer tool outputs the selected parcel.

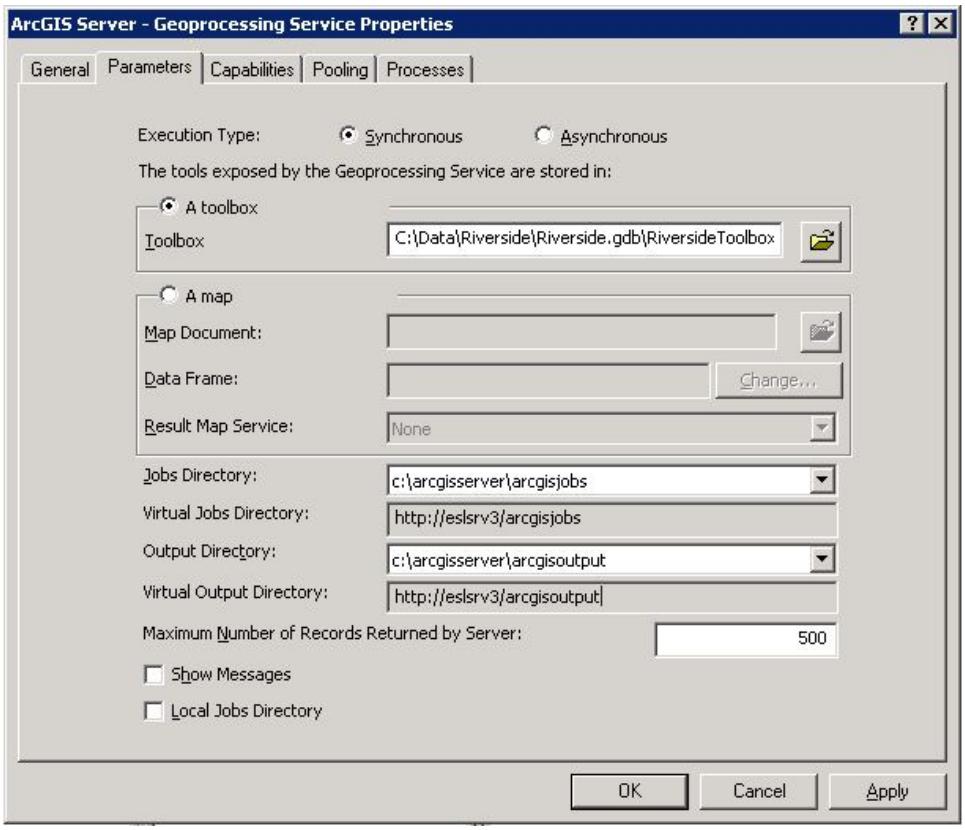
  At the same time, the Parcels data layer is also input into another Make Feature Layer tool that outputs a parcel layer that will be used by the Select Layer By Location tool in step 3.

- **Step 2**—The Buffer tool takes the selected parcel input and creates a 300-foot buffer around it. The output buffer polygon is sent to the Select Layer By Location tool.

- **Step 3**—The Select Layer By Location tool selects parcel features that are within the 300-foot buffer from the parcel layer created in step 1. It exposes them as a GPFeatureRecordSetLayer parameter that can be used to render the surrounding parcels in the map display and/or create a list of them.

Throughout the model, selections are used, so ArcGIS Server does not have to save information to disk. This reduces disk I/O, which translates into a more efficient geoprocessing service. Also consider (1) the use of in_memory workspace in geoprocessing models to reduce disk I/O and (2) publishing them as synchronous services.*

Properties of the Geoprocessing Service



The Python version of the model is presented below for further details:

```
# Import relevant modules
import sys, os, arcgisscripting, traceback

# Create the geoprocessing scripting object
gp = arcgisscripting.create()

try:
  # Get Input argument...
  APN = gp.GetParameterAsText(0)
```

* More information on tuning geoprocessing services can be found in the ArcGIS Server Help topic Performance tips for geoprocessing services.

```
# Local variables...
  Parcels = "<path to your parcels>"
  Parcel_Layer = "Parcels_Layer"
  Selected_Parcel = "Selected_Parcels_Layer"
  Buffer_Output = "in_memory\\Parcels_Buffer"

  # Make a Feature Layer for the parcel specified
  gp.MakeFeatureLayer_management(Parcels, Selected_Parcel,
"\"APN\" = '%APN%'", "",
"AREA AREA VISIBLE NONE;APN APN VISIBLE NONE;CITY_OWNED
CITY_OWNED VISIBLE NONE;METROSCAN METROSCAN VISIBLE
NONE;Shape_Length Shape_Length VISIBLE NONE;Shape_Area
Shape_Area VISIBLE NONE")

   # Buffer the specified parcel
   gp.Buffer_analysis(Selected_Parcel, Buffer_Output, "300
   Feet", "FULL", "ROUND", "NONE", "")

   # Make a Feature Layer of the parcels
   gp.MakeFeatureLayer_management(Parcels, Parcel_Layer,
"", "", "AREA AREA VISIBLE NONE;APN APN VISIBLE
NONE;CITY_OWNED CITY_OWNED VISIBLE NONE;METROSCAN METROSCAN
VISIBLE NONE;Shape_Length Shape_Length VISIBLE
NONE;Shape_Area Shape_Area VISIBLE NONE")

   # Select all the parcels that intersect the buffer
   gp.SelectLayerByLocation_management(Parcel_Layer,
"INTERSECT", Buffer_Output, "", "NEW_SELECTION")

   gp.SetParameterAsText(1, Parcel_Layer)
   gp.SetParameterAsText(2, Selected_Parcel)
   gp.SetParameterAsText(3, Buffer_Output)

except:
   tb = sys.exc_info()[2]
   tbinfo = traceback.format_tb(tb)[0]
   pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo +
     "\nError Info:\n " + \
   str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
   gp.AddError(pymsg)

   msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
   gp.AddError(msgs)
```

Appendix E: Map Queries and the Geometry Service

This appendix discusses the workflow of using the Riverside Street Map to search for a parcel, buffer the found parcel boundary, and generate a report of parcels falling within that buffer.

The code below illustrates the above workflow but with U.S. state data available from an ArcGIS℠ Online sample server.

Finding a State

The find task (findTask) is executed with parameters defined by "myFindParams". The function findTaskCompleteHandler is called after successful execution of the task.

```
<!-- Find Task -->
<esri:FindTask id="findTask"
executeComplete="findTaskCompleteHandler(event)"
   url="http://sampleserver1.arcgisonline.com/ArcGIS/rest/
services/Specialty/ESRI_StatesCitiesRivers_USA/MapServer/
find" />

<!-- Parameters for Find Task -->
<esri:FindParameters id="myFindParams"
       returnGeometry="true"
       contains="true"
       searchText="{fText.text}"
       layerIds="[2]"
       searchFields="['STATE_ABBR','STATE_NAME']" />
<!-- Find Task -->
 private function
findTaskCompleteHandler(event:FindEvent):void
 {
      myGraphicsLayer.clear();
       // add only first feature as graphic to graphics
      layer, ignore remaining
       var graphic:Graphic = event.findResults[0].feature;
       graphic.toolTip = event.findResults[0].
       foundFieldName + ": " +
 event.findResults[0].value;
       myGraphicsLayer.add(graphic);
       findResGraphic = graphic;
       // zoom to extent of all features
       map.extent = Polygon(event.findResults[0].
       feature.geometry).extent.expand(1.1);
       doBuffer();
 }
```

Buffer Found State Using Geometry Service

The buffer of the found state is performed by executing the function doBuffer. The buffer is executed on the geometry service defined by "myGeometryService" using the appropriate parameters set in the doBuffer method. On completion of the buffer operation, the function bufferCompleteHandler is called, which adds the buffered geometry to the graphics layer.

```
<!-- Geometry Service -->
<esri:GeometryService id="myGeometryService"
 url="http://sampleserver2.arcgisonline.com/ArcGIS/rest/
services/Geometry/GeometryServer"/>

private function doBuffer():void
    {
       var bufferParameters:BufferParameters = new
       BufferParameters();
       bufferParameters.features = [findResGraphic];
       bufferParameters.distances = [bText.text];
       bufferParameters.unit = BufferParameters.UNIT_FOOT;
       bufferParameters.bufferSpatialReference = new
       SpatialReference(4326);


myGeometryService.addEventListener(GeometryServiceEvent.BUF
FER_COMPLETE, bufferCompleteHandler);
 myGeometryService.buffer(bufferParameters);
    }

    private function bufferCompleteHandler
   (event:GeometryServiceEvent):void
    {

      myGeometryService.removeEventListener(GeometryService
Event.BUFFER_COMPLETE, bufferCompleteHandler);
       bufferResGraphic = event.graphics[0];
       myGraphicsLayer.add(bufferResGraphic);
       doQuery();
    }
```

Report Generation

To generate the report, a query task is executed using the buffered geometry as the spatial extent to find all the enclosing states. The attributes of the selected states are used to populate the data grid.

```
<!-- Query Task for generating report -->
<esri:QueryTask id="queryTask"
url="http://sampleserver1.arcgisonline.com/ArcGIS/rest/
services/Specialty/ESRI_StatesCitiesRivers_USA/MapServer/2"
/>
```

```
private function doQuery():void
{
var query:Query = new Query();
query.geometry = bufferResGraphic.geometry;
query.returnGeometry = true;
query.spatialRelationship = "esriSpatialRelIntersects";
query.outFields=["STATE_NAME","STATE_FIPS","SUB_REGION","ST
ATE_ABBR","POP1990","POP1999"];
queryTask.execute(query);
   }

<mx:DataGrid id="resultsGrid" width="100%"
dataProvider="{queryTask.executeLastResult.attributes}"
visible="{queryTask.executeLastResult != null}" >
   <mx:columns>
      <mx:DataGridColumn headerText="State Name"
dataField="STATE_NAME"/>
      <mx:DataGridColumn headerText="Region"
      dataField="SUB_REGION"/>
      <mx:DataGridColumn headerText="FIPS"
      dataField="STATE_FIPS"/>
      <mx:DataGridColumn headerText="Abbreviation"
dataField="STATE_ABBR"/>
      <mx:DataGridColumn headerText="Population 1990"
dataField="POP1990"/>
      <mx:DataGridColumn headerText="Population 1999"
dataField="POP1999"/>
   </mx:columns>
</mx:DataGrid>
```

The complete source code for executing this workflow appears below:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
   xmlns:mx="http://www.adobe.com/2006/mxml"
   xmlns:esri="http://www.esri.com/2008/ags">
   <mx:Script>
<![CDATA[
import com.esri.ags.tasks.Query;
import com.esri.ags.SpatialReference;
import com.esri.ags.tasks.BufferParameters;
import com.esri.ags.events.GeometryServiceEvent;
import com.esri.ags.geometry.Polygon;
import com.esri.ags.Graphic;
import com.esri.ags.events.FindEvent;

      private var findResGraphic : Graphic;
      private var bufferResGraphic : Graphic;
```

```
   <!-- Find Task -->
   private function findTaskCompleteHandler
   (event:FindEvent):void
   { myGraphicsLayer.clear();
   // Add only first feature as graphic to graphics layer,
   ignore remaining
   var graphic:Graphic = event.findResults[0].feature;
   graphic.toolTip = event.findResults[0].foundFieldName +
 ": " + event.findResults[0].value;
   myGraphicsLayer.add(graphic);
   findResGraphic = graphic;
   // zoom to extent of all features
   map.extent =
   Polygon(event.findResults[0].feature.geometry).extent.ex
   pand(1.1);
   doBuffer();
}

private function doBuffer():void
{
   var bufferParameters:BufferParameters = new
   BufferParameters();
   bufferParameters.features = [findResGraphic];
   bufferParameters.distances = [bText.text];
   bufferParameters.unit = BufferParameters.UNIT_FOOT;
   bufferParameters.bufferSpatialReference = new
   SpatialReference(4326);

    myGeometryService.addEventListener(GeometryServiceEvent.
BUFFER_COMPLETE, bufferCompleteHandler);
    myGeometryService.buffer(bufferParameters);
}

   private function
   bufferCompleteHandler(event:GeometryServiceEvent):void
{
      myGeometryService.removeEventListener(GeometryService
Event.BUFFER_COMPLETE, bufferCompleteHandler);
      bufferResGraphic = event.graphics[0];
      myGraphicsLayer.add(bufferResGraphic);
      doQuery();
}

   private function doQuery():void
{
      var query:Query = new Query();
      query.geometry = bufferResGraphic.geometry;
      query.returnGeometry = true;
      query.spatialRelationship =
"esriSpatialRelIntersects";
```

```
      query.outFields=["STATE_NAME","STATE_FIPS","SUB_REGIO
      N","STATE_ABBR","POP1990","POP1999"];
         queryTask.execute(query);
}
]]>
</mx:Script>
   <!-- Find Task -->
   <esri:FindTask id="findTask"
   executeComplete="findTaskCompleteHandler(event)"

   url="http://sampleserver1.arcgisonline.com/ArcGIS/rest/
   services/Specialty/ESRI_StatesCitiesRivers_USA/MapServer/
   find" />

   <!-- Parameters for Find Task -->
   <esri:FindParameters id="myFindParams"
      returnGeometry="true"
      contains="true"
      searchText="{fText.text}"
      layerIds="[2]"
      searchFields="['STATE_ABBR','STATE_NAME']" />

   <!-- Geometry Service -->
   <esri:GeometryService id="myGeometryService"

   url="http://sampleserver2.arcgisonline.com/ArcGIS/rest/
   services/Geometry/GeometryServer"/>

   <!-- Query Task for generating report -->
   <esri:QueryTask id="queryTask"
   url="http://sampleserver1.arcgisonline.com/ArcGIS/rest/
   services/Specialty/ESRI_StatesCitiesRivers_USA/
   MapServer/2"/>

      <mx:Panel width="100%" height="100%">
         <mx:HBox>
         <mx:Label text="State Name" />
      <mx:TextInput id="fText" text="Neva" />
      </mx:HBox>
      <mx:HBox>
         <mx:Label text="Buffer distance" />
         <mx:TextInput id="bText" text="5" />
      </mx:HBox>

      <mx:Button label="Find"
      click="{findTask.execute(myFindParams)}" />

   <esri:Map id="map">
   <esri:ArcGISTiledMapServiceLayer
   url="http://server.arcgisonline.com/ArcGIS/rest/
   services/ESRI_StreetMap_World_2D/MapServer"/>
```

```
<esri:GraphicsLayer id="myGraphicsLayer" />
</esri:Map>

<mx:DataGrid id="resultsGrid" width="100%"
dataProvider="{queryTask.executeLastResult.attributes}"
visible="{queryTask.executeLastResult != null}" >
   <mx:columns>
       <mx:DataGridColumn headerText="State Name"
       dataField="STATE_NAME"/>
       <mx:DataGridColumn headerText="Region"
       dataField="SUB_REGION"/>
       <mx:DataGridColumn headerText="FIPS"
       dataField="STATE_FIPS"/>
       <mx:DataGridColumn headerText="Abbreviation"
       dataField="STATE_ABBR"/>
       <mx:DataGridColumn headerText="Population 1990"
       dataField="POP1990"/>
       <mx:DataGridColumn headerText="Population 1999"
       dataField="POP1999"/>
       </mx:columns>
       </mx:DataGrid>

</mx:Panel>

</mx:Application>
```

Appendix F: Testing Methodology and Definitions

A Web mapping application is a popular way to distribute GIS content to many users over the Web. With this paradigm comes the need for the mapping application to communicate with the server over the Web through HTTP, a common Web protocol.

To adequately model a user accessing the Web application, common activities and interactions need to be written down and reproduced using the Web application within a Web browser. While the user activity is being performed, a tool that is able to record all the Web traffic being passed between the server and client is needed to capture the transactions. The captured traffic is subsequently replayed through a performance test tool capable of simulating multiple concurrent requests appearing as different users.

For this use case, the testing tool was Microsoft Visual Studio Team Systems 2008 Test Edition (VSTS). Within VSTS, the recorded Web traffic can be edited to support the dynamic input of query parameters. This enables the application of tests on different areas of the dataset as well as examination of what multiple users in different areas, performing different actions, would do to the system. Other functionalities available within VSTS are the complete Performance Monitor (Perfmon) integration, which provides insight into Windows-based servers with regard to CPU, disk, RAM, and so forth, usage during the different stages of testing.

When performing tests with a test tool as complex as VSTS, the definition of users and threads is an important concept to understand:

- **User:** An intelligent consumer of data services within the rich Internet application that requires a pause between interactions to decide if further interaction is needed. The pause between user interactions is described as a think time.

- **Thread:** A consumer of data services within the rich Internet application that requires no pause between interactions to decide if further interaction is needed.

While testing the Riverside Viewer Web mapping application, two different approaches were used, and each illustrates something different. First, the user-based approach is a request made by the test tool with a defined wait time between recorded requests to simulate an intelligent user interaction. This is common in capacity planning exercises to effectively determine if a capacity requirement can be met. The second approach is the thread-based approach, common in a benchmark scenario to show performance differences between services.

About Esri

Since 1969, Esri has been helping organizations map and model our world. Esri's GIS software tools and methodologies enable these organizations to effectively analyze and manage their geographic information and make better decisions. They are supported by our experienced and knowledgeable staff and extensive network of business partners and international distributors.

A full-service GIS company, Esri supports the implementation of GIS technology on desktops, servers, online services, and mobile devices. These GIS solutions are flexible, customizable, and easy to use.

Our Focus

Esri software is used by hundreds of thousands of organizations that apply GIS to solve problems and make our world a better place to live. We pay close attention to our users to ensure they have the best tools possible to accomplish their missions. A comprehensive suite of training options offered worldwide helps our users fully leverage their GIS applications.

Esri is a socially conscious business, actively supporting organizations involved in education, conservation, sustainable development, and humanitarian affairs.

Contact Esri

1-800-GIS-XPRT (1-800-447-9778)
Phone: 909-793-2853
Fax: 909-793-5953
info@esri.com
www.esri.com

Offices worldwide
www.esri.com/locations

380 New York Street
Redlands, CA 92373-8100 USA

Printed in USA