Rocket®

EBOOK

Top 5 Myths of Green-Screen Modernization

The Evolution of Application Modernization Technology

Introduction

Imagine that it’s your first day at a new job. You’re working for a company that relies on timely and accurate access to customer information. You might be a claims processor at an insurance company, a customer service representative at a bank, or a sales manager at a car dealership. One of the first tasks your supervisor gives you is to spend your initial three weeks on the job learning a business-critical computer application that looks like it’s from 1975. How would that make you feel?

Surprisingly, many organizations still ask their employees to use green-screen (or text-based) user interfaces to work with essential business information, typically because these systems, while venerable, still perform critical tasks for the organization. If you manage employees who use these applications, you recognize that while green-screen applications do important work, they’re hard to learn (even intimidating) for unfamiliar workers, and often they’re only accessible with terminal emulation software. In addition, these applications are closed to the outside world: They can’t make use of application programming interfaces (APIs) from third-party vendors that would improve the user experience, and green-screen application functionality can not be accessed by other, more modern web and mobile applications.

Green screens are both a business and a technology problem. How can a business stay competitive when it takes three weeks to train employees on green-screen applications? How can a group of customer service representatives help customers on the phone when they must navigate through several green-screen applications simultaneously to implement the customer’s request? How can a business share select critical business functionality with employees and partners, or allow for sharing with other applications, when that functionality is not accessible for external application integration?

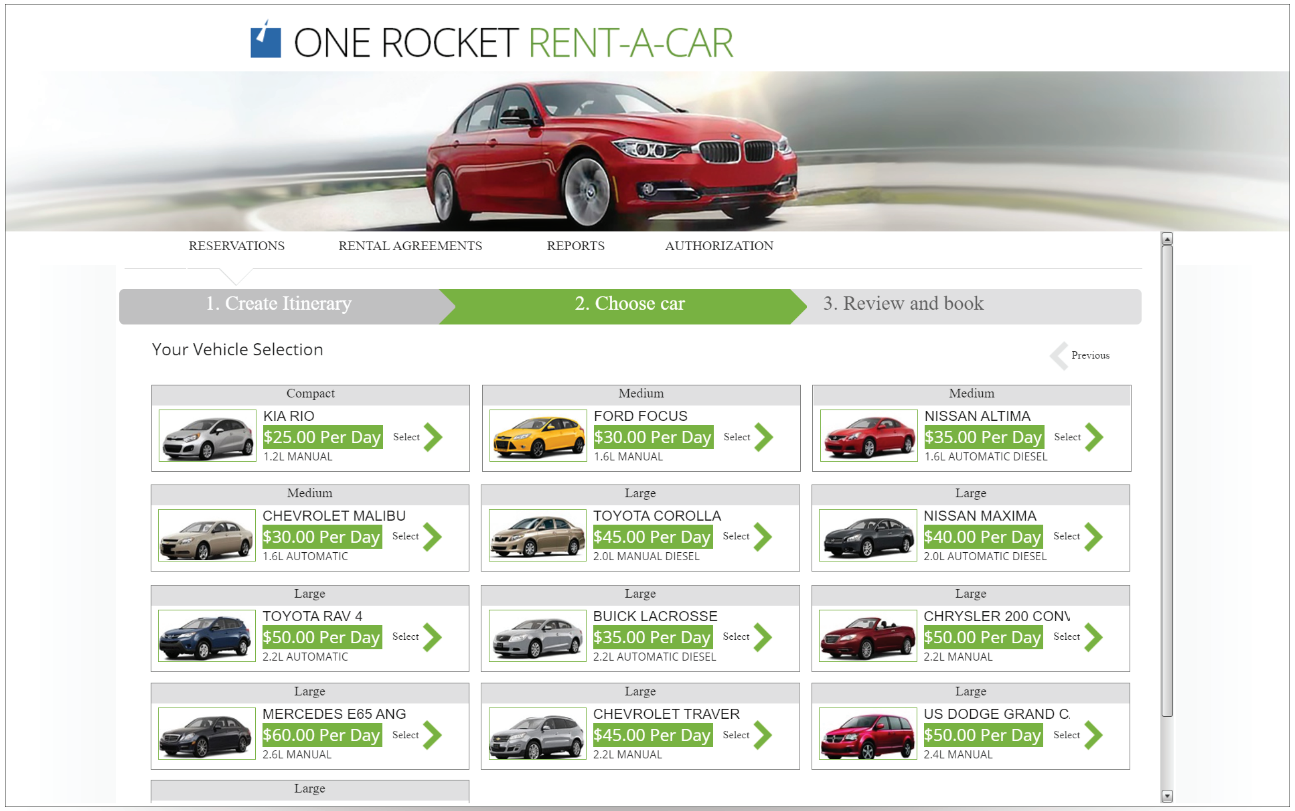
From a technical standpoint, the problem is simply that green-screen applications are no longer the de facto standard for user interfaces. Host-based applications run businesses, because they’re reliable, secure, and they house decades of business rules and information. Application developers need a way to help businesses use the information and business logic from these systems, but deliver it in a modern and flexible way.

Getting Past the Myths

There are ways to turn green-screen applications into more current and useful web-based applications, or allow for sharing with other applications. These methods offer value by building on the logic embedded in green-screen applications, rather than requiring source-code changes or screen-scraping.

Modernization approaches that use the application logic can be vastly more powerful, efficient, and easier to maintain than screen-scrapers. This paper is intended to debunk some common myths about modernization technologies, and demonstrate what advanced modernization strategies can enable you to do compared with more traditional approaches.

Myth #1: Modernization Solutions are Just Green Screens in Browsers



ONE ROCKET RENT-A-CAR

RESERVATIONS RENTAL AGREEMENTS REPORTS AUTHORIZATION

1. Create Itinerary | 2. Choose car | 3. Review and book

Your Vehicle Selection

Previous

Compact KIA RIO $25.00 Per Day Select 1.2L MANUAL	Medium FORD FOCUS $30.00 Per Day Select 1.6L MANUAL	Medium NISSAN ALTIMA $35.00 Per Day Select 1.6L AUTOMATIC DIESEL
Medium CHEVROLET MALIBU $30.00 Per Day Select 1.6L AUTOMATIC	Large TOYOTA COROLLA $45.00 Per Day Select 2.0L MANUAL DIESEL	Large NISSAN MAXIMA $40.00 Per Day Select 2.0L AUTOMATIC DIESEL
Large TOYOTA RAV 4 $50.00 Per Day Select 2.2L AUTOMATIC	Large BUICK LACROSSE $35.00 Per Day Select 2.2L AUTOMATIC DIESEL	Large CHRYSLER 200 CONV $50.00 Per Day Select 2.2L MANUAL
Large MERCEDES E65 ANG $60.00 Per Day Select 2.6L MANUAL	Large CHEVROLET TRAVER $45.00 Per Day Select 2.2L MANUAL	Large US DODGE GRAND C $50.00 Per Day Select 2.4L MANUAL
Large		

Customized interfaces give users the same host information with much improved functionality.

Myth 1

A basic requirement for green-screen modernization is the ability to dynamically display a GUI version of any application screen. This typically involves setting up generic templates or rules that interpret each screen on the fly and present a GUI version of the screen to the user. Dynamic GUI is useful for deploying applications to desktops or the web, but adds little functional value from the user or business perspective. It is sometimes referred to as "green screen in a browser."

The real value-add in screen-based modernization projects lies in the ability to rework the user experience to match today's business needs. Users need improved application usability and workflow, plus integration with other desktop- and web-based tools they require for their jobs. Screens still form the basis for a customized GUI; however, the screens behave more as application-level APIs than direct user interfaces.

While some older approaches focus on "green screen in a browser," current modernization technologies combine information from multiple green screens in addition to the data on the current screen, and present it to the user for a useful and efficient application workflow.

Myth #2: Keeping Host and GUI in Sync is Hard

As any developer knows, maintaining APIs can be a labor-intensive process. If an API changes, anything that uses that API must typically also be adapted. So, if a green screen changes, how do you take that change into account in your GUI?

Managing host application change is a key differentiator between current modernization technologies and a typical screen-scraping approach. An effective modernization solution uses your application screen maps[1] during application development and maintenance. Using a repository of screen maps as a basis, developers can create and maintain customized GUIs for applications that include thousands of screens. When screens change, an automated change management feature enables you to easily synchronize your GUI application with your underlying screen "APIs."

Host-to-GUI synchronization is actually a two-part challenge:

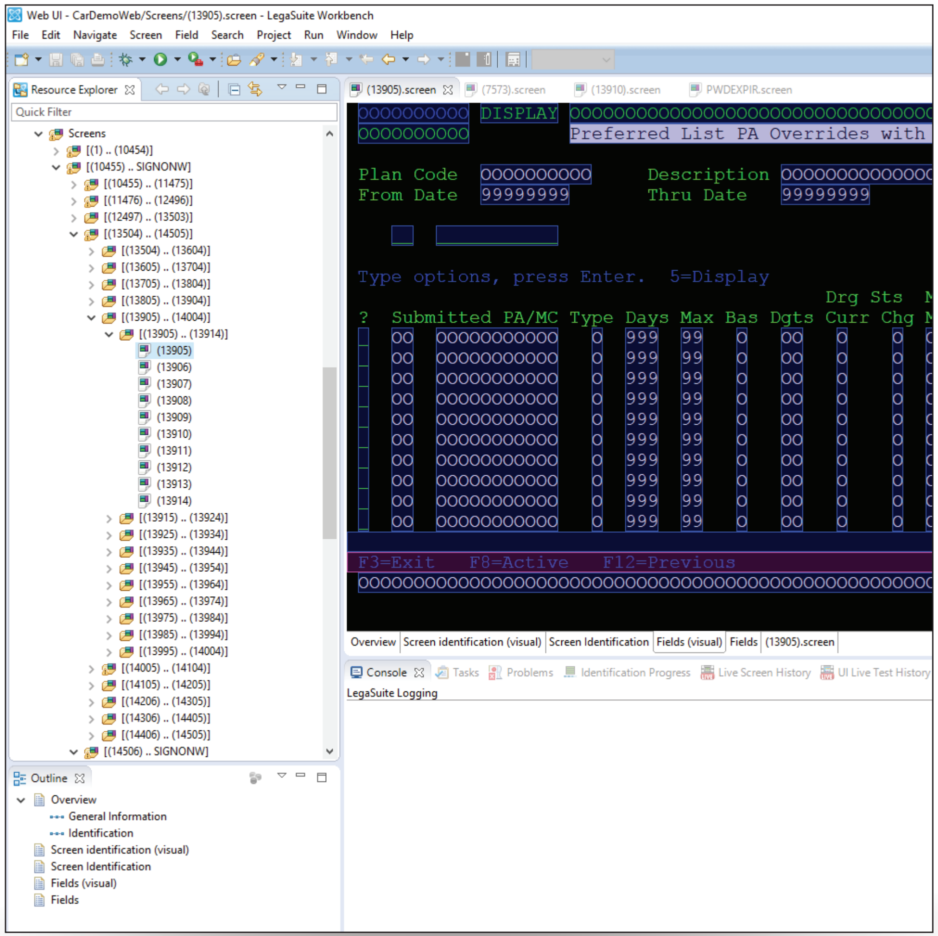
- The GUI must know what screen(s) to use to access information
- The GUI must know what host fields to use on each screen

Traditional approaches expect you to manually maintain the links between your live green screens and your GUI. If a screen changes, there is no automated way to catch and apply the change. By registering your application screen map files in an XML-based repository, today's modernization solutions overcome these synchronization challenges and provide a complete foundation for application development and maintenance.

Myth 2

[1] Mainframe applications typically use screen map types such as BMS and MFS. System i applications typically use display files (DSPF).

Rocket.



```
Screens
  [(1) .. (10454)]
  [(10455) .. SIGNONW]
    [(10455) .. (11475)]
    [(11476) .. (12496)]
    [(12497) .. (13503)]
    [(13504) .. (14505)]
      [(13504) .. (13604)]
      [(13605) .. (13704)]
      [(13705) .. (13804)]
      [(13805) .. (13904)]
      [(13905) .. (14004)]
        [(13905) .. (13914)]
          (13905)
          (13906)
          (13907)
          (13908)
          (13909)
          (13910)
          (13911)
          (13912)
          (13913)
          (13914)
        [(13915) .. (13924)]
        [(13925) .. (13934)]
        [(13935) .. (13944)]
        [(13945) .. (13954)]
        [(13955) .. (13964)]
        [(13965) .. (13974)]
        [(13975) .. (13984)]
        [(13985) .. (13994)]
        [(13995) .. (14004)]
      [(14005) .. (14104)]
      [(14105) .. (14205)]
      [(14206) .. (14305)]
      [(14306) .. (14405)]
      [(14406) .. (14505)]
    [(14506) .. SIGNONW]
```

```
Overview
  General Information
  Identification
Screen identification (visual)
Screen Identification
Fields (visual)
Fields
```

```
OOOOOOOOOOO  DISPLAY  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOO           Preferred List PA Overrides with

Plan Code  OOOOOOOOOOO     Description  OOOOOOOOOOOOOOOO
From Date  99999999        Thru Date    99999999



Type options, press Enter.  5=Display
                                                Drg Sts  M
?  Submitted PA/MC Type Days Max Bas Dgts Curr Chg  M
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O
   OO   OOOOOOOOOOO   O  999  99   O   OO    O    O   O

F3=Exit   F8=Active   F12=Previous
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

As shown in this repository, the ability to keep track of screens is a key difference between screen-scraping and modernizing.

What screen am I on?

Custom GUIs for green-screen applications require a link between the GUI and the host screens that provide information. In other words, the GUI solution must “know” what screens are being sent by the host to show the right GUIs.

Many products provide manual screen identification that requires developers to navigate to each desired screen in the live application and manually select the characteristics that make that screen unique. If the developer accidentally selects a conditional field, or uses the same characteristics to identify two different screens, identification can fail. If a screen changes, the developer must manually navigate to the screen again and re-identify it.

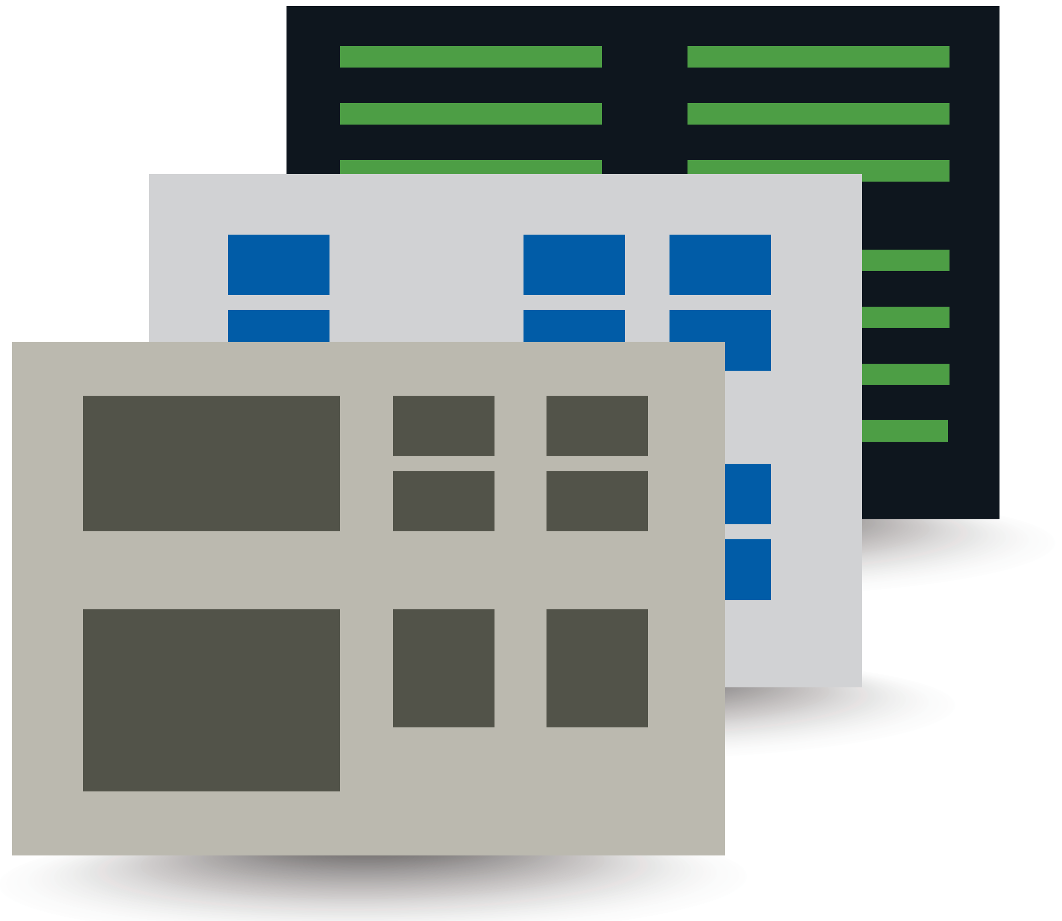
More advanced modernization solutions automatically maintain screen identifications. Screen maps are incorporated into a repository, and are automatically assigned unique identifiers. Since this process is based on the screen maps, rather than the live data stream, it's more accurate. The links between GUIs and the green screens in the repository are automatically checked and updated whenever the change-management process is initiated. Automated screen identification ensures an accurate link between your GUI and your host applications—without requiring the developer to spend any time maintaining these critical links.

What fields do I need?

As users work with your GUI, information they enter or change is pushed back to editable fields on the green screen. Therefore, GUIs built over green-screen applications require a link between the host fields on each screen that provide the information and GUI fields that enable users to interact with that information.

A typical approach to addressing this challenge is to rely on the developer to create manual mappings based on the row/column position of each host field on each screen. Developers navigate live to each screen that contains the fields they want, then point and click and click on each individual field to register it for use in the GUI. Solution developers may not know that certain fields change color, size, or position based on program events, and therefore cannot take these changes into account. They may be unaware of conditional fields that only display in certain circumstances. Without access to the screen maps, these solutions place the burden on the developer to discover changes in field states and attributes. A more precise method is clearly necessary.

A truly effective modernization solution offers automatic host field mapping based on the application field names and attributes from the screen maps. Information about every host field in the application becomes available through the screen repository. When


Rather than using row/column positions, an effective Web-enablement tool should offer automatic host field mapping

developers have access to this information, they can create solutions that encompass all the host application possibilities—including features such as detecting and acting upon color and state changes and variable field positions and sizes.

Using the true application field names also reduces change management requirements, since the modernization solution interacts with fields based on their true host names, rather than their row and column positions on the screen. If you move a field on the screen, a modern solution can still interact with it without changing the underlying field definition.

Myth #3: Most Modernization Solutions Are Maintenance Nightmares

One of the most pervasive myths about modernization solutions is that they create maintenance nightmares. This myth is well-earned, because maintenance is a genuine challenge with traditional solutions. Because these solutions scrape live screens for development, they offer no automated way to accommodate for—or even be aware of—host application changes. This shifts the maintenance burden to the developer, who is expected to identify and react to these changes. This reactive approach often means that maintenance is not initiated until after users have already experienced a problem.

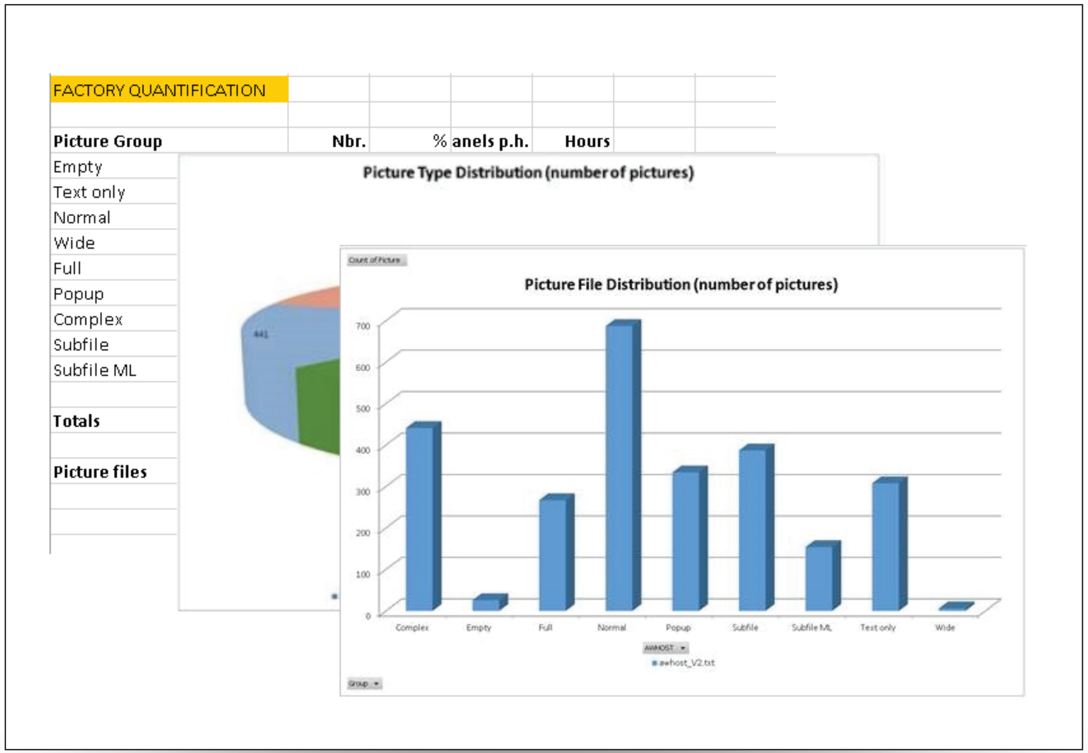
As we've already touched upon, a modernization solution should feature change management technology designed to keep application screens and corresponding GUIs synchronized. This is a critical point—accurate change management reduces your application maintenance burden and elevates the quality and reliability of your modernized solution.

When a solution uses a screen repository to manage host-to-GUI and host-to-API screen changes, ongoing maintenance becomes a highly automated process. Developers can proactively schedule synchronization with the host application screens at any time. The synchronization process updates the screen repository with the latest screen maps; makes corresponding updates to screen identifications, host field information, and the GUI; and produces a complete

Myth 3

report of all activities. These processes can all run unattended (in batch mode), or can be performed step-by-step by individual developers. Work is typically limited to adjusting the layout of the customized GUI to accommodate added fields on the host, and customizing GUIs for new host screens. You can also manage working files with common source control tools such as SYN or Git.

Myth #4: Modernization Projects are Unmanageable



FACTORY QUANTIFICATION			
Picture Group	**Nbr.**	**% anels p.h.**	**Hours**
Empty			
Text only			
Normal			
Wide			
Full			
Popup			
Complex			
Subfile			
Subfile ML			
Totals			
Picture files			

An effective modernization solution gives you visibility into your project.

Some tools are used to quickly complete a UI project at the last minute. Because of this perception, project management is often not considered until after a solution is identified. This can be a costly mistake. With these tactical approaches, developers operate in the dark. When building a new GUI front-end solution based only on live screens, visibility into the level of effort is limited to the developer's knowledge of the live application. You may be unaware of application features that only display in certain circumstances, or of modifications that only apply to certain customers or business units. Your visibility into maintenance costs is also limited.

However, if you're using a modernization solution that provides you with complete information about all the screens that are part of your project, you'll be able to clearly understand, estimate, and document the amount of work required to complete a modernization project. If you are considering a screen-based web enablement project, you will have applications in mind that you want to convert. How many screens are in those applications? If you don't know how many screens require GUIs, how can you estimate the time and effort it will take to complete the project? How will you even know when the project is complete?

A complete modernization solution should use a screen repository to let you quantify the work required to complete a project, and offer tools that let you ascertain the exact number of screens in any application or module, as well as the complexity of each screen. A project estimator feature should also enable you to assign time-to-complete values for screens of varying complexity, and document the level of effort down to individual screens and fields, to successfully implement your new user experience.

Myth #5: You Can't Use APIs with Modernization Solutions

Some modernization solutions offer the ability to turn business processes within existing enterprise applications into discrete APIs. For example, an API might take an existing application function, such as "look up a customer address" or "add an item to an order," and encapsulate it into a RESTful service. The service is then made available to other applications within or outside the organization. When other applications need information from the host application, they call the required API and receive a reply through web services. The underlying host application remains unchanged.

While this is a different type of modernization for an existing application, all the challenges that apply to creating a new GUI also apply to creating new services. Traditional approaches still don't address the fundamental challenges: automating service mapping to the right screens and fields, synchronizing host changes with the services that rely on these screens and fields, and being able to accurately estimate the level of effort required to create and maintain services. Speed is another issue. Can traditional approaches meet service-level requirements? This is a valid concern for poorly architected solutions.

To work in high-volume environments that require frequent changes to business logic, a service-enablement solution must be capable of busting all the myths presented in this eBook. It should also execute services independently of any GUI modernization layer, to ensure maximum

Myth 5

Painless Screen-Based Modernization and Integration

If you need a modernization solution, but are wary of development hurdles or the quality of the result, Rocket® LegaSuite software will change your outlook. Rocket LegaSuite and Rocket® API offer a complete web- and API-enablement solution to modernize your existing text-based application portfolio.

If you're planning a web- or service-enablement project for an existing host-based application—whether it's based on a mainframe, IBM i, OpenVMS, or UNIX-VT platform—consider the advantages of Rocket solutions:

- Synchronized development and maintenance for screens and their corresponding GUIs and services
- Automated screen identification and management
- Accurate host-to-GUI and host-to-service field mapping based on true field names rather than row/column positioning
- Detailed project planning and management capabilities
- Comprehensive support for IBM® i and heterogeneous application lifecycle management through Rocket® Aldon Lifecycle Manager

Rocket®

Rocket Modernization Solutions

Whether you've used another solution in the past or are just beginning to explore the possibilities of web- and service-enablement for your existing applications, we invite you to experience the alternatives offered by Rocket Software:

- Rocket LegaSuite enables you to web-enable critical back-end applications and access them from web and mobile browsers. It makes web development fast and easy, with drag-and-drop tools and runtime components that extend and repurpose any host-based application as a user-friendly, HTML5 web application.

- Rocket API enables you to unlock the value of your green-screen application logic for access from virtually any web or mobile application, so you can ensure outstanding user experiences that mirror the way that your stakeholders do business. Using Rocket Rocket API, you wrap functionality within existing critical business systems in APIs that can be integrated with new or existing cloud, mobile, and self-service applications—without any risky code changes to your existing systems.

Here's how to determine which Rocket solution best suits your needs:

What You're Trying to Do	Rocket Solution(s)
Create a modern web or mobile interface for host-based applications on the IBM i, mainframe, OpenVMS, or UNIX-VT platforms, optionally including third-party APIs	Rocket LegaSuite
Create a new web or mobile application that draws functionality from multiple host-based applications	Rocket API and Rocket LegaSuite
Access functionality from host-based applications for use with external web, mobile, or other applications	Rocket API

To find out more about Rocket LegaSuite and Rocket API, please contact us today for a no-obligation, personal demonstration.

EBOOK

Rocket Software (www.rocketsoftware.com) is a technology company that helps organizations in the IBM ecosystem build solutions that meet today's needs while extending the value of their technology investments for the future. Thousands of companies depend on Rocket to solve their most challenging business problems by helping them run their existing infrastructure and data, as well as extend those assets to take advantage of cloud, mobile, analytics, and other future innovations. Founded in 1990, Rocket is based in Waltham, Massachusetts with locations in Europe, Asia, and Australia.

rocketsoftware.com
info@rocketsoftware.com
US: 1 855 577 4323 EMEA: 0800 520 0439 APAC: 612 9412 5400
twitter.com/rocket
www.linkedin.com/company/rocket-software
www.facebook.com/RocketSoftwareInc
blog.rocketsoftware.com

Rocket®

© Rocket Software, Inc. or its affiliates 1990 – 2017. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates. 201709EBFMOGSMV2