

Path not found

Disparities in Access to Computer Science Courses in California High Schools



Level Playing Field Institute

Path Not Found

Disparities in Access to Computer Science Courses in California High Schools

Report by Alexis Martin, Frieda McAlear, and Allison Scott

Copyright © May 2015

Cover Photo: Gabriela Nemeth 2012

Design by Design Action Collective



Level Playing Field Institute

2201 Broadway, Suite 101, Oakland, CA 94612.

PHONE: 415-946-3030

FAX: 415-946-3001

EMAIL: info@lpfi.org

WEBSITE: www.lpfi.org

 [@lpfi](https://twitter.com/lpfi)

 facebook.com/levelplayingfield

LPFI MISSION STATEMENT: Level Playing Field Institute is committed to eliminating the barriers faced by underrepresented people of color in science, technology, engineering and math (STEM) and fostering their untapped talent for the advancement of our nation.

ACKNOWLEDGEMENTS: *The authors would like to thank the following individuals for assistance with extensive data acquisition and report feedback: Gary Page, Karl Scheff, Dan Lewis, Randy Bonnell, Eugene Lemon, Jane Margolis, Joanna Goode, Dan Garcia, David Bernier, Solomon Russell, Gail Chapman, Julie Flapan, Debra Richardson, Emmanuel Onyeador, Claire Shorall, Jean Wing, and Bryan Twarek. We would also like to thank the Kapor Center for Social Impact, as well as the individual, corporate, governmental, and foundation sponsors who provide support for the Level Playing Field Institute's research and programming.*

Foreword

This report is released in the wake of national protests against discriminatory police actions that devalue the lives of young people of color. For the first time in more than a generation, mass mobilizations have brought the daily struggles of these young men and women to the halls of the Department of Justice and to the center of our national conversation.

At the same time, communities of color have developed digital-age tactics to grapple with inequities in public safety, housing, healthcare, education, and employment. Digital media tools such as the #BlackLivesMatter hashtag highlight injustices in the experiences of people of color in the United States. Programs like [Hands Up United Tech Impact](#) have begun providing computer science exposure for youth of color in Ferguson and beyond, and have created opportunities for underrepresented groups to become producers of new technologies, not merely consumers.

Since 2001, the Level Playing Field Institute has worked to identify and eliminate the barriers faced by underrepresented populations in science, technology, engineering and mathematics. Our work takes on a greater urgency today. Last fall, for the first time in history, students of color made up the majority of first graders nationwide.

Given this backdrop, disparities in computer science education in public schools will not only widen the opportunity and achievement gap, they threaten to have a profound negative economic impact as technology takes on an increasingly central role in the economy. *Path Not Found* exposes these disparities in California public high schools and presents a roadmap for lawmakers, educational programs, the tech community, and school districts to make a crucial course correction.

Benjamin Todd Jealous

Board Member, Level Playing Field Institute

Partner, Kapor Capitol



Photo © Liz Acosta 2012.

Executive Summary

Twenty-first century careers and economic growth in the United States are increasingly dependent upon computing expertise. In California, the home of Silicon Valley, the economy is tied to the sustainability of its rapidly-growing technology sector. Unfortunately, diversity statistics among leading Silicon Valley technology companies indicate that the technology workforce is overwhelmingly white and male, while women and people of color are greatly underrepresented relative to their proportion of the population. With the changing racial landscape of the state and the nation, the lack of diversity within computing fields suggests there is a large pool of untapped talent which comprises a critical component of the future computing workforce.

Given the rising demand for skilled computer science professionals in California, it is vital that the state's public schools provide all students with a solid

foundation in computer science coursework. However, California's school system is failing to prepare its students—particularly low income students and students of color—for the technology jobs of the future.

Path Not Found exposes one of the foundational causes of underrepresentation in computing: disparities in access to computer science courses in California's public high schools. The Level Playing Field Institute conducted analyses to disaggregate current computer science offerings by student demographic variables. This report illuminates vast disparities in access to computer science courses in California public high schools by race, socioeconomic status, and linguistic background, and finds that computer science courses are offered at consistently higher rates in schools with student populations that are already disproportionately represented in the computing sector.

This report contains several key findings:

The availability of AP Computer Science courses is considerably higher in schools with lower populations of underrepresented students of color. Further, the higher a school's percentage of underrepresented students of color, the lower the likelihood of a school offering any computer science courses whatsoever.

- Schools with the highest percentage of underrepresented students of color offer computer science courses at a rate nearly **half** that of schools with the lowest percentage of underrepresented students of color.
- Schools with the highest percentage of underrepresented students of color offer AP Computer Science at a rate **twelve times lower** than schools with the lowest percentage of underrepresented students of color.

California public high schools with high percentages of low-income students have overwhelmingly fewer opportunities to take computer science courses.

- Schools with the highest percentage of low-income students offer computer science courses at a rate **less than half** that of schools with the lowest percentage of low-income students.
- Schools with the highest percentage of low-income students offer AP Computer Science at a rate nearly **eleven times** lower than schools with the lowest percentage of low-income students.

Disparities in computer science course availability can also be seen when examining the percentage of English learners within a school's student population.

- Just **8%** of schools with a high percentage of English learners (11% or above) offer AP Computer Science. In contrast, **19%** of schools with a low percentage of English learners (10% and below) offer AP Computer Science.
- Just **31%** of schools with a high percentage of English learners (11% or above) offer any Computer Science courses. In contrast, **39%** of schools with a low percentage of English learners (10% and below) offer any computer science courses.

Computer science course availability is also low within the largest California schools districts, most of which serve high populations of students of color and low-income students.

- **10** out of the largest **20** districts in California do not offer AP Computer Science.
- **5** out of the largest **20** districts in California do not offer any computer science courses.
- Of the 560,874 high school students in the largest 20 California districts, just **1%** (8,136) are enrolled in any computer science course.

In a region at the forefront of technological innovation, opportunities to join the fastest-growing industries must be available to all students regardless of race, ethnicity, primary language, or socioeconomic status.

Reducing disparities in access to computer science coursework requires state, district, school, and community-level funding and policy commitments. This report suggests the following recommendations:

- 1 Develop state-level and district-level funding strategies to create equitable access to both introductory and Advanced Placement computer science coursework across all California public high schools.
- 2 Ensure all California school districts allow computer science to count as either a mathematics or science high school graduation requirement.
- 3 Develop a statewide shared definition of what courses constitute “computer science” for use in all California high schools, in order to create consistency as well as transparency in access.
- 4 Ensure introductory computer science courses provide the necessary scaffolding and effective instruction for students of all backgrounds to succeed in advanced computing coursework.
- 5 Ensure computer science curricula, pedagogy, and assessments are culturally-relevant and inquiry-based in order to engage underrepresented groups and broaden participation in computer science.
- 6 Expand and strengthen the state’s computing teacher workforce by adopting recently-proposed modifications to California’s computing-related supplementary authorization so that fully credentialed teachers in subjects other than mathematics can teach computer science with the proper training and preparation.
- 7 Expand and institutionalize regional partnerships between technology companies and California high schools, to capitalize on the prevalence of computer science professionals who can serve as volunteer instructors, mentors, guest speakers, or classroom teaching assistants (from underrepresented backgrounds when possible).
- 8 Expand access to in-school and out-of-school programs designed to develop computing interest among underrepresented groups, particularly through hands-on projects, field trips, extracurricular activities, and mentorship programs. Ensure funding prioritizes programs serving low-income students of color and other underrepresented groups.

Introduction

“No other subject will open as many doors in the 21st century, regardless of a student’s ultimate field of study or occupation, as computer science.”

—Computer Science Teachers Association, *Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age*

The stakes for broadening participation in computing and the technology sector have never been higher. Computing and technology occupations continue to be among the highest-paying and fastest-growing occupations, growing twice as fast as the average rate for all fields.¹ Recent technological advances driving the state and national economy across many industries can be linked to the field of computer science. Economic projections indicate that there will be more than 1.3 million job openings in computing and mathematical occupations by 2022.² In California, the home of Silicon Valley—where the economy is tied to the sustainability of its technology sector—rapid job creation and growth in productivity are increasingly present within computing industries.

Simultaneously, recent releases of employment diversity statistics among leading Silicon Valley technology companies (situated in one of the most diverse states in the nation) indicate that the technology workforce is overwhelmingly white and male, while women and people of color are greatly underrepresented relative to their proportion of the population.³ This underrepresentation is concerning for several reasons. Research has demonstrated that diverse groups and teams are associated with increased success and innovation,⁴ which not only impacts continued economic success of the technology industry but also has implications for creating

innovative technological solutions to tackle major societal problems affecting all segments of society. Additionally, with the rapidly changing racial landscape of the state and the nation, the lack of diversity within the computing fields suggests there is a large pool of untapped talent which comprises a critical component of the future computing workforce. Finally, the underrepresentation of diverse groups within the technology industry greatly affects future economic opportunities for communities of color.

Path Not Found exposes one of the foundational causes of this underrepresentation: disparities in access to computer science courses in California’s public high schools. Previous reports⁵ have demonstrated disparities in participation on the Advanced Placement (AP) Computer Science exam or have examined A–G computer science courses⁶ that may not actually be currently taught in schools. The Level Playing Field Institute expanded on these prior analyses by utilizing data on the range of computer science courses currently being taught in California public high schools and conducting analyses to disaggregate access to courses by student demographic variables. Hence, this report illuminates vast disparities in access to computer science courses in California public high schools by race, socioeconomic status, and linguistic background.⁷ *Path Not Found* concludes with a description of promising practices

- 1 Lacey, Alan, and Benjamin Wright. 2010. “Occupational Employment Projections to 2018.” *Monthly Labor Review*, US Bureau of Labor Statistics. US Bureau of Labor Statistics. 2013. “Employment Projections Program: Employment by Detailed Occupation.”
- 2 Richards, Emily, and David Terkanian. 2015. “Occupational Employment Projections to 2022.” *Monthly Labor Review*, US Bureau of Labor Statistics.
- 3 Gilpin, Lyndsey. 2015. “Diversity in Tech: 10 Data Points You Should Know.” *Tech Republic*, February 4, 2015.
- 4 Hunt, Vivian, Dennis Layton, and Sara Prince. 2015. “Why Diversity Matters.” McKinsey & Company, January.
- 5 Bernier, D., Chris Stephenson, Debra Richardson, and Gail Chapman. 2012. “In Need of Repair: The State of Computer Science Education in California.” California Computing Education Advocacy Network, January.
- 6 California STEM Learning Network. 2014. “Computer Science Education in California.” September.
- 7 University of California Office of the President approves a set of high school courses, known as “A–G” requirements, that students must complete to be minimally eligible for admission to the University of California and California State University systems. Approval of a course put forth by a particular school, however, does not guarantee that the course will be offered.
- 8 It is important to note that throughout this report, “access” is defined as course availability. However, research has shown that even if a computer science course is offered at a school, it may not, in fact, be accessible for many populations of students due to scheduling constraints or lack of teacher/counselor guidance. For more on access issues beyond the availability of courses, see: Margolis, Jane, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kim Nao. 2008. *Stuck in the shallow end: Education, race, and computing*. Cambridge: MIT Press.

and a set of statewide recommendations to promote equitable access to computer science for all students. By addressing these disparities in access to computer science courses, more equitable pathways to

the fastest-growing industries will be created, thus providing growth and sustainability opportunities for communities of color, the technology industry, and the nation's economy.

Underrepresentation in Computer Science

Given the rising demand for skilled computer science professionals in California, it is vital that the state's public schools provide a solid foundation in computer science coursework. Yet, California's school system is failing to prepare its students—particularly low income students and students of color—for the technology jobs of the future. Across the state, **65%** of public high schools offer **no** computer science courses. Further, only **13%** of California public high schools offer the AP Computer Science course, and just 6,676 of the state's 1.95 million high school-aged students (**.03%**) took the AP Computer Science exam in 2014.⁸ AP Computer

Science is critical to exposing and preparing students to major in computer science in college, and research indicates that without access to advanced computer science courses in high school, students are eight times less likely to pursue computer science in higher education.⁹ Despite the fact that African American and Latino students comprise a combined 59% of the high-school aged population in California,¹⁰ a combined total of only 731 African American and Latino students took the AP Computer Science exam in 2014, representing just 11% of the state's total AP Computer Science test-taking population.¹¹

FIGURE 1
Percentage of AP Computer Science Test-takers and Statewide HS Population, by Race/Ethnicity

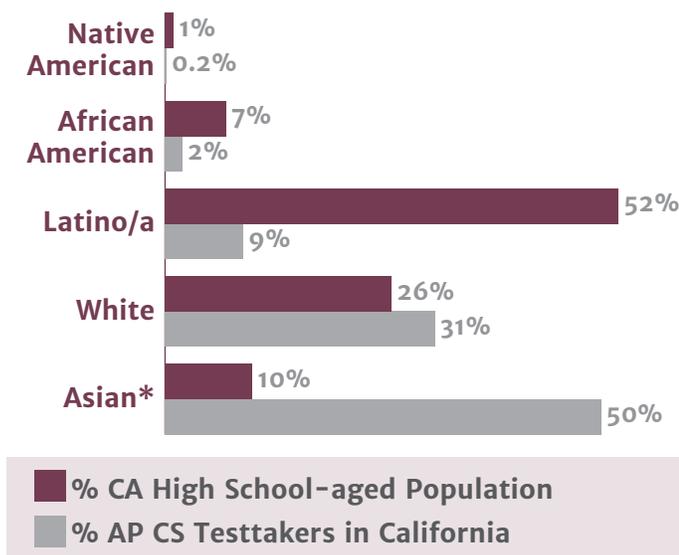
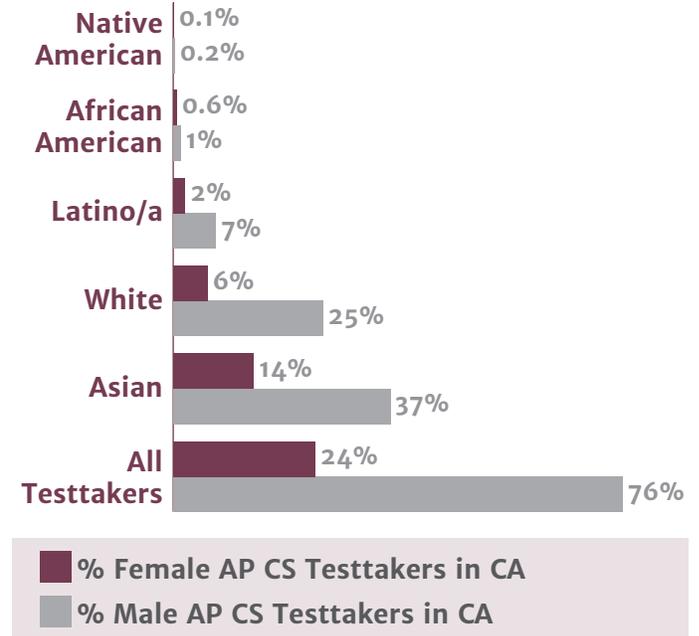


FIGURE 2
Percentage of AP Computer Science Test-takers, by Gender and Race/Ethnicity



* California Department of Education demographic data for Asian students aggregated with Pacific Islanders and Filipino students

8 College Board. 2014. "AP Program Participation and Performance Data, State Report: California." California Department of Education. 2015. "Fingertip Facts on Education in California - CalEdFacts."
 9 Mattern, Krista, Shaw, Emily, and Ewing, Maureen. 2011. *Advanced Placement Exam Participation*. 6th ed. College Board.
 10 California Department of Education, Educational Demographics Unit. 2014. "Statewide Enrollment by Ethnicity."
 11 College Board. 2014. "AP Program Participation and Performance Data, State Report: California."

Research consistently indicates that women and people of color are severely underrepresented among those taking computer science courses, pursuing and completing computer science Bachelor's, Master's and Doctorate degrees, and those participating in the computing workforce. In post-secondary education, African American and Latino students combined account for just 17% of all computer science Bachelor's degrees conferred, 7% of all computer science Ph.D.'s conferred, 6% of Computer Science faculty, and ultimately just 9% of the computing workforce nationwide.¹²

Numerous causes of underrepresentation in

computer science have been identified, including: lack of access to rigorous computer science courses,¹³ lack of engaging and culturally relevant computing curriculum,¹⁴ lack of diverse role models and peer networks,¹⁵ negative racial and gender stereotypes about ability,¹⁶ implicit bias and unwelcoming classroom, lab, and workplace environments.¹⁷ While acknowledging the multifaceted barriers affecting participation in computer science among underrepresented groups, this report addresses the fundamental barrier of access to computer science coursework within California's public high schools.

Disparities in Access to Computer Science Courses

Detailed analyses of California Department of Education course, school, and district data¹⁸ revealed that access to computer science courses in California public high schools significantly varies by student

demographics, with the majority of schools with high populations of underrepresented students of color¹⁹ and/or low-income students significantly less likely to offer computer science courses.²⁰

Race/Ethnicity

As illustrated in Figure 3, while only a fraction of California's public high schools offer AP Computer Science, the availability of these courses is much higher in schools with lower populations

of underrepresented students of color. Further, the higher a school's percentage of underrepresented students of color, the lower the likelihood of a school offering any computer science courses whatsoever.

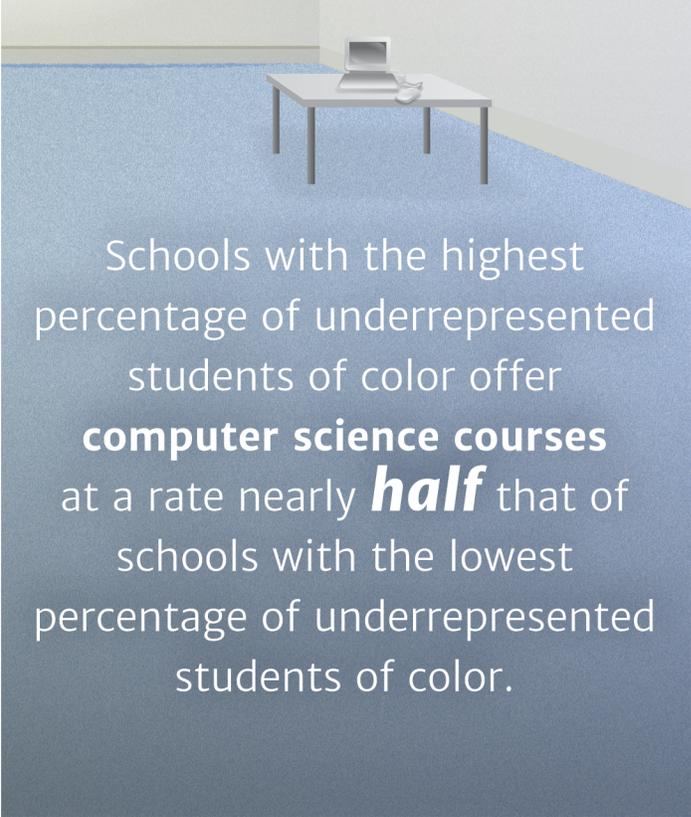
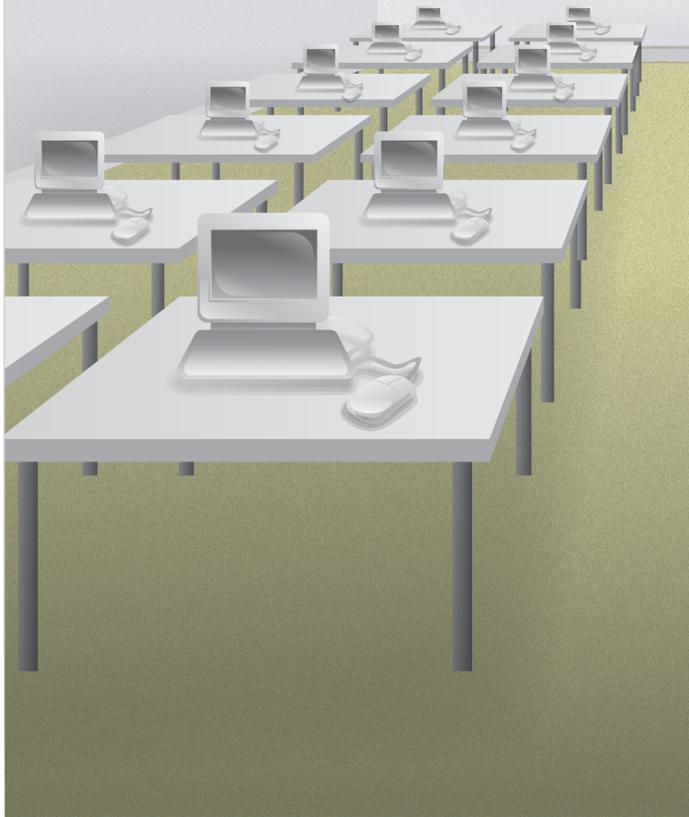
- 12 National Science Foundation, National Center for Science and Engineering Statistics. 2015. "Women, Minorities, and Persons with Disabilities in Science and Engineering."
- 13 Margolis, Jane, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kim Nao. 2008. *Stuck in the shallow end: Education, race, and computing*. Cambridge: MIT Press.
- 14 Ryoo Jean, Jane Margolis, Clifford Lee, Cueponcaxochitl Sandoval, and Joanna Goode. 2013. "Democratizing computer science knowledge: transforming the face of computer science through public high school education." *Learning, Media and Technology*. 38(2), 161-181.
- 15 Scott, Kimberly, Gregory Aist, and Denice Hood. 2009. "CompuGirls: Designing a Culturally Relevant Technology Program." *Educational Technology*, 49(6), 34-39.
- 16 Cain, Curtis. 2012. Underrepresented Groups in Gender and STEM: The Case of Black Males in CISE. *Proceedings of the 50th annual conference on Computers and People Research*, 97-102.
- 17 Zimmerman, Thomas, David Johnson, Cynthia Wambsgans, and Antonio Fuentes. 2011. "Why Latino high school students select computer science as a major: Analysis of a success story." *ACM Transactions on Computing Education*. 11(2).
- 18 Aronson, Joshua and Claude Steele. 1995. "Stereotype Threat and the Intellectual Test Performance of African-Americans." *Journal of Personality and Social Psychology*, 69(5), 797-811.
- 19 Cheryan, Sapna, Paul Davies, Victoria Plaut, and Claude Steele. 2009. "Ambient belonging: How stereotypical cues impact gender participation in computer science." *Journal of Personality and Social Psychology*, 97(6), 1045-1060.
- 20 Level Playing Field Institute analyzed California Basic Educational Data System/California Longitudinal Pupil Achievement Data System computer science course offerings for the 2013-14 school year, provided by the California Department of Education, as well as publicly available [school/district demographic data](#) to produce this report. With the exception of alternative/continuation schools, and schools with fewer than 100 students, all California public high schools were included in analyses. This analysis is based on the most accurate data available, though there may be reporting errors from schools or districts.
- 19 Defined as percentage of student body that is African American, Latino/a, and/or Native American; while disparities exist within Asian and Pacific Islander populations, there is not sufficient data to disaggregate by subgroups within these categories. Percentage cutoffs were adapted from the [UCLA Civil Rights Project](#) definition of school segregation.
- 20 Throughout this report, the term "any computer science course" refers solely to courses with either "computer science" or "computer programming" in the title (includes: Computer Science, Computer Programming, Advanced Placement Computer Science, Computer Operations Science, and Exploring Computer Science), in order to focus on the academic discipline of computer science. Computer science does not include the often conflated computer-based courses on information technology and computer [literacy and usage](#) (e.g., Networking, Information Technology, Animation, Computer Literacy/Lab, among others). This was informed by the Computer Science Teachers Association's [definition of Computer Science Education](#). See also: [CSTA Computer Science Standards](#)

FIGURE 3

Computer Science Availability by Underrepresented Student Population

Computer science course *availability is considerably lower* in California public high schools that have *high populations of underrepresented students of color*.

Schools with the highest percentage of underrepresented students of color offer **AP Computer Science** at a rate *twelve times lower* than schools with the lowest percentage of underrepresented students of color.



Schools with the highest percentage of underrepresented students of color offer **computer science courses** at a rate nearly *half* that of schools with the lowest percentage of underrepresented students of color.

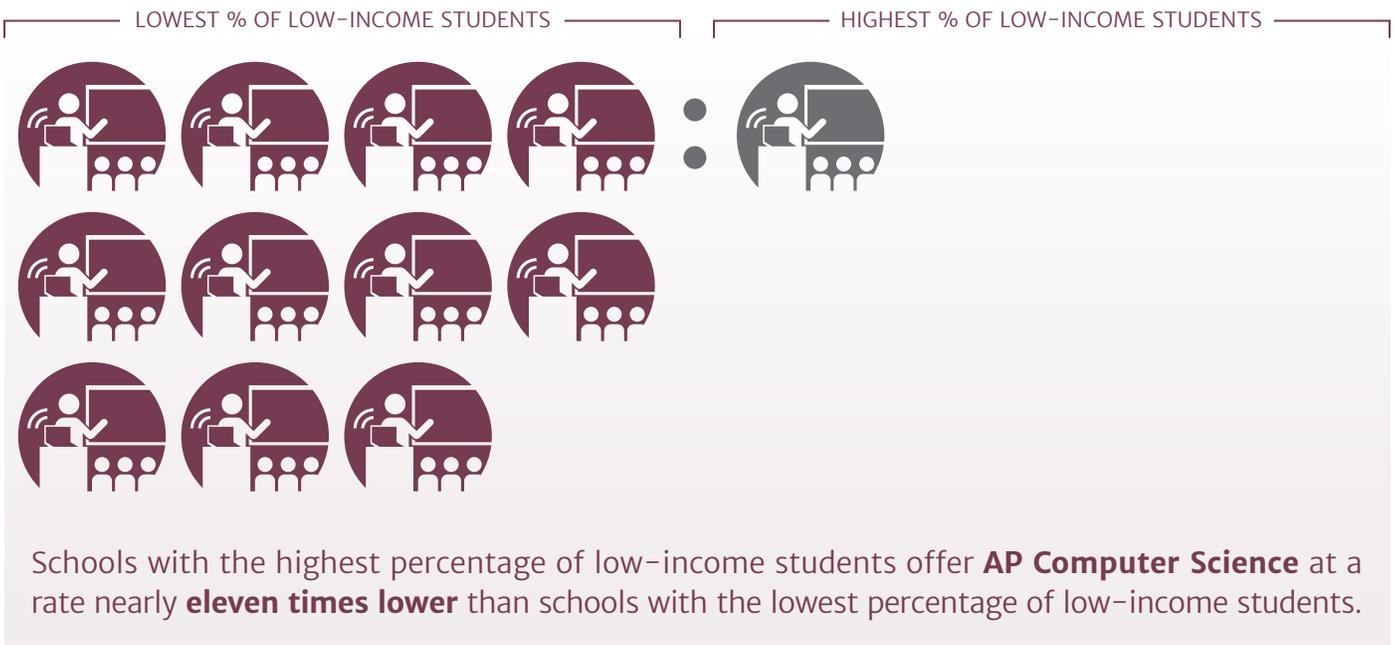
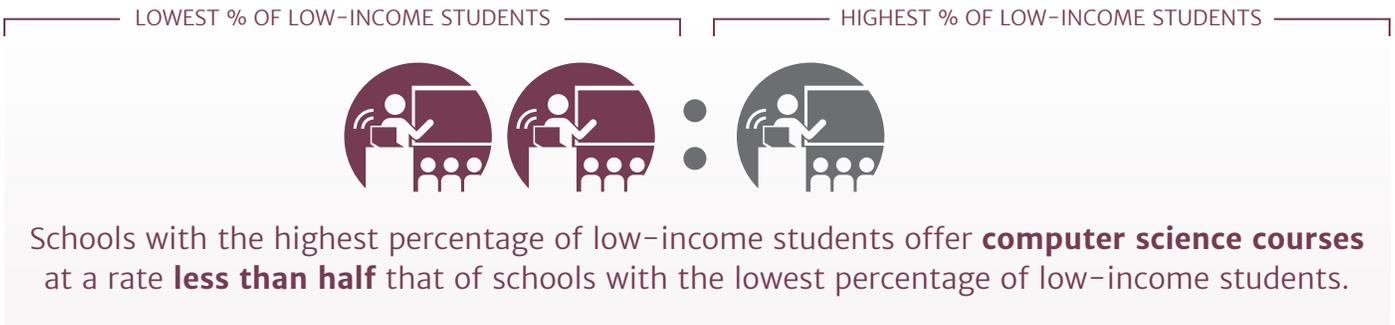
BY THE NUMBERS...

Percentage Underrepresented Students of Color in Total Student Body	Number of CA Public High Schools	Number and Percent of schools offering AP Computer Science		Number and Percent of schools offering Any Computer Science	
0-50%	523	126	24%	233	45%
51-90%	513	44	9%	144	28%
91-100%	248	5	2%	66	27%

Income

The socioeconomic status of a school's student population is also associated with access to computer science courses. California public high schools with high percentages of low-income students²¹ have overwhelmingly fewer computer science opportunities (Figure 4).

FIGURE 4
Computer Science Availability by Percentage of Low-Income Students



BY THE NUMBERS...

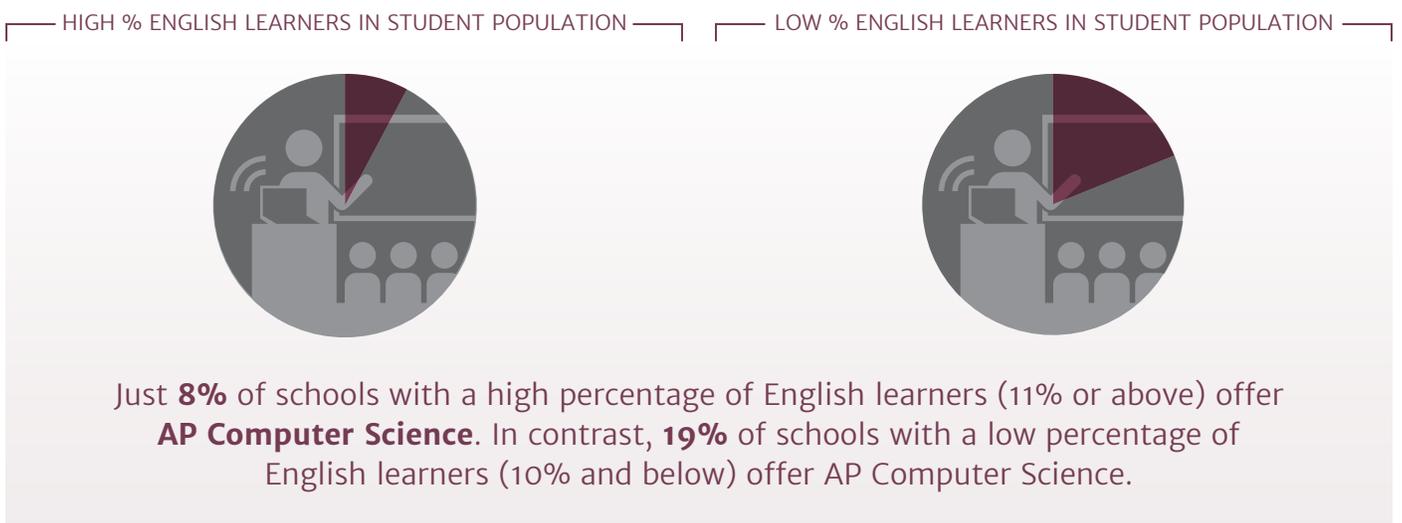
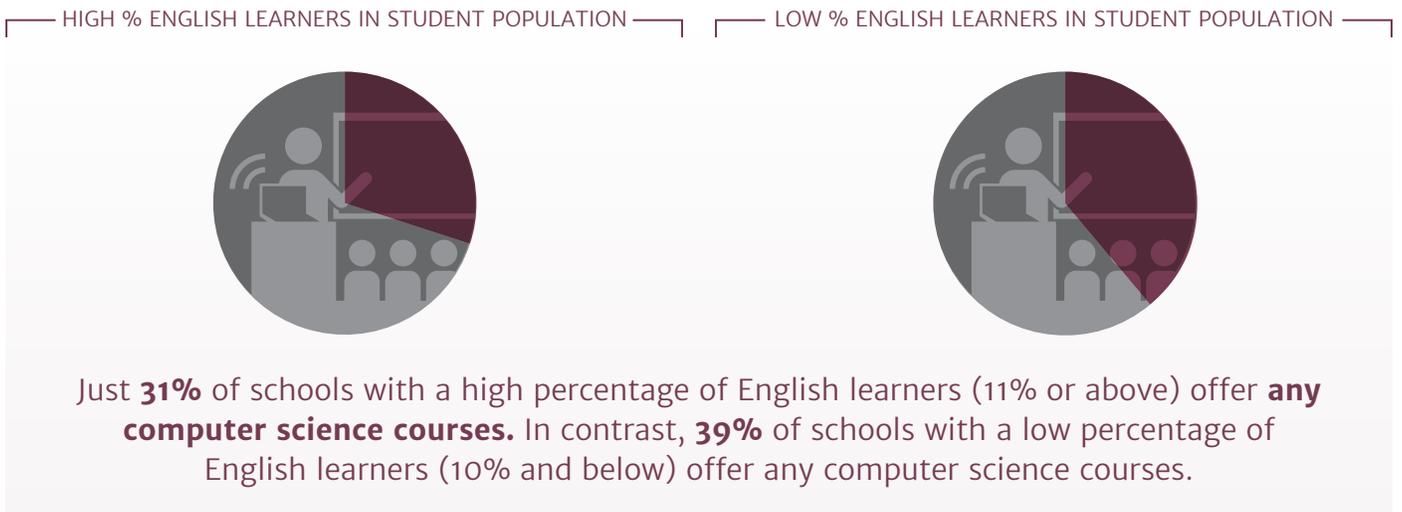
Percentage Low Income Students in Total Student Body	Number of CA Public High Schools	Number and Percent of schools offering AP Computer Science		Number and Percent of schools offering Any Computer Science	
1-25%	198	85	43%	120	61%
26-50%	305	43	14%	101	33%
51-75%	403	33	8%	130	32%
76-100%	378	14	4%	92	24%

21 As defined by Free/Reduced Price Lunch eligibility (through federally-determined poverty guidelines) for the National School Lunch Program.

Language Learners

Disparities in computer science course availability can also be seen when examining the percentage of English learners²² within a school's student population (Figure 5).

FIGURE 5
Computer Science Availability by English Learner Status



BY THE NUMBERS...

Percentage English Learners in Total Student Body	Number of CA Public High Schools	Number and Percent of schools offering AP Computer Science		Number and Percent of schools offering Any Computer Science	
0-10%	671	260	39%	129	19%
11% or more	613	187	31%	46	8%

²² "English learner" students, as defined by the California Department of Education, have a primary language other than English on the state-approved Home Language Survey and lack the defined English language skills of listening comprehension, speaking, reading, and writing necessary to succeed in traditional instructional programs.

Computer Science in California's Largest Districts

Table 1 lists student enrollment in computer science courses for the largest California schools districts, most of which serve high populations of students of color and low-income students. Though these districts educate a combined total of **over one-quarter (29%)** of California's high school-aged students, they offer few opportunities for students to access computer science coursework.

- **10 out of the largest 20** districts in California do not offer AP Computer Science.
- **5 out of the largest 20** districts in California do not offer any computer science courses.
- The district with the highest percentage of underrepresented students of color in Table 1 (Santa Ana; 96% underrepresented students of color) has **zero students** enrolled in any computer science courses.
- Of the 560,874 high school students in the largest 20 California districts, just **1%** (8,136) are enrolled in any computer science course.

TABLE 1
Computer Science Course Enrollment²³ in Largest California School Districts²⁴

District Name	High School Enrollment	% Free/Reduced Price Lunch Student Population	% Under-represented Students of Color	# of Students Enrolled in AP Computer Science	# of Students Enrolled in any other Computer Science course ²⁵	Total % of Students in District Enrolled in Computer Science courses
Los Angeles Unified	198,180	79%	83%	166	6,131 ²⁶	3%
San Diego Unified	38,549	67%	55%	160	3	.04%
Kern High School District	37,086	54%	69%	0 ²⁷	0	0% ²⁷
Sweetwater Union High	28,947	55%	77%	0 ²⁷	60	.02% ²⁷
Long Beach Unified	26,103	67%	70%	51	0	.01%
Fresno Unified	21,057	92%	75%	23	66	.04%
Elk Grove Unified	19,405	60%	42%	0	152	.07%
San Francisco Unified	18,548	64%	33%	136	351	3%
Corona-Norco Unified	17,521	44%	57%	38	0	.02%
Capistrano Unified	17,273	24%	26%	60	36	.05%
Santa Ana Unified	16,838	91%	96%	0	0	0%
San Bernardino City Unified	15,601	94%	88%	0	206	1%
San Juan Unified	16,411	50%	28%	0	0	0%
Garden Grove Unified	14,881	70%	54%	67	0	.04%
Riverside Unified	13,803	64%	66%	1	172	1%
Sacramento City Unified	13,038	68%	54%	0	87	.06%
Fontana Unified	12,863	87%	91%	0	0	0%
Clovis Unified	12,624	42%	36%	0	0	0%
Oakland Unified	12,096	77%	69%	28 ²⁸	72 ²⁸	.08%
Stockton Unified	10,050	83%	75%	0	70	.06%

23 Most recent available enrollment data, from 2012-13 school year.

24 Districts in California with the top 20 overall student enrollment, arranged by largest high school population. Source: California Department of Education. (2014). "District Enrollment by Grade."

25 See footnote 20 for description of included courses.

26 Includes 2014-15 Exploring Computer Science (ECS) enrollment numbers, obtained from ECS program.

27 District has added one AP Computer Science course since this data was released, though enrollment numbers are not yet available.

28 2014-15 enrollment numbers, obtained from OUSD Research and Evaluation Department.

Conclusion and Recommendations

In addition to the broad lack of access to computer science courses affecting all students in California's public high schools, significant inequities in access to computer science exist by student demographics. In a region at the forefront of technological innovation, opportunities to join the fastest-growing industries must be available to all students regardless of race, ethnicity, primary language, or socioeconomic status. Democratizing access to computer science knowledge, however, is more than just an imperative to improve economic and workforce outlooks. Computing is truly a 21st century skill; computer

science exposure—when rooted in culturally relevant instruction and rigorous standards—has the potential to foster critical inquiry and develop problem solving abilities that transcend the study of computer science and are highly relevant for all fields of study.²⁹

Broad and coordinated efforts are needed to reverse disparities in computer science access, particularly for underrepresented students of color and low-income students across the state of California. To address disparities in computer science opportunity, this report highlights promising practices and suggests the following recommendations:

- 1 Develop state-level and district-level funding strategies to create equitable access to both introductory and Advanced Placement computer science coursework across all California public high schools.
- 2 Ensure all California school districts allow computer science to count as either a mathematics or science high school graduation requirement.
- 3 Develop a statewide [shared definition](#) of what courses constitute “computer science” for use in all California high schools, in order to create consistency as well as transparency in access.
- 4 Ensure introductory computer science courses provide the necessary scaffolding and effective instruction for students of all backgrounds to succeed in advanced computing coursework.
- 5 Ensure computer science curricula, pedagogy, and assessments are [culturally-relevant](#) and [inquiry-based](#) in order to engage underrepresented groups and broaden participation in computer science.
- 6 Expand and strengthen the state’s computing teacher workforce by adopting [recently-proposed modifications](#) to California’s computing-related supplementary authorization so that fully credentialed teachers in subjects other than mathematics can teach computer science with the proper training and preparation.
- 7 Expand and institutionalize regional partnerships between technology companies and California high schools, to capitalize on the prevalence of computer science professionals who can serve as volunteer instructors, mentors, guest speakers, or classroom teaching assistants (from underrepresented backgrounds when possible).
- 8 Expand access to in-school and out-of-school programs designed to develop computing interest among underrepresented groups, particularly through hands-on projects, field trips, extracurricular activities, and mentorship programs. Ensure funding prioritizes programs serving low-income students of color and other underrepresented groups.

29 Goode, Joanna, Jane Margolis, and Gail Chapman. 2014. “Curriculum Is Not Enough: The Educational Theory and Research Foundation of the Exploring Computer Science Professional Development Model.” In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*: 493–98.

Promising Practices

A number of programs and initiatives across California—both in and out of schools—aim to address gaps in access to computer science courses and provide opportunities for exposure and engagement in computer science.³⁰

District and School-Level Initiatives

- Los Angeles Unified School District partners with the [Exploring Computer Science](#) program to provide curricula and professional development to educators teaching the year-long Exploring Computer Science course at nearly 40 LAUSD high schools, exposing over 2,300 high school students—the majority of whom are from groups traditionally underrepresented in computer science—to an engaging and culturally relevant computer science curriculum. This program has achieved strong [outcomes](#) and has impacted student interest in computer science.
- Los Angeles Unified School District has also begun a [partnership with Code.org](#) in order to offer computer science to grades K–8 and to expand high school computer science course offerings. Code.org additionally partners with a number of districts and schools across the state of California.
- San Francisco Unified School District also partners with Code.org to broaden computer science access. Additionally, SFUSD is [crafting an initiative](#) to make computer science compulsory for all students in grades Pre–K through 8 and to expand computer science opportunities at all district high schools.
- [Oakland Unified School District](#), in partnership with [Level Playing Field Institute](#), co-founded a Computer Science Working Group comprised of administrators, non-profit partners, and district math, science, and computer science teachers. This group worked collaboratively to assess computer science educational assets and challenges, to pilot computer science professional development programs, and to offer specific recommendations aimed at strengthening computer science offerings throughout the district.
- In order to create new AP Computer Science courses, teachers at Castlemont High School in Oakland and Lincoln High School in San Jose undertook [successful crowdfunding campaigns](#) to buy laptops and equipment, and partnered with the [Technology Education And Literacy in Schools](#) program to bring tech professionals into classrooms as volunteer instructors.
- California chapters of the [Computer Science Teachers Association](#) work to develop strong communities of computer science teachers. The organization supports the teaching of computer science and provides opportunities for K–12 teachers and students.

Out-of-School Opportunities

- [Level Playing Field Institute](#) implements computer science initiatives in Northern and Southern California designed to provide underrepresented students opportunities for exposure, engagement, and technical skill development within the field of computer science. These programs include National Science Foundation-funded rigorous computer science coursework for 9–12th grade students in SMASH (Summer Math and Science Honors Academy), computer science exposure for African–American middle school boys in SMASH: Prep, and Hackathons to increase exposure and “Level the Coding Field” for 6–12th grade students.
- [Black Girls Code](#) provides girls of color with opportunities to learn skills in computer programming through workshops, after-school programs, and Hackathons.
- [Teens Exploring Technology](#) offers programs for young men of color to learn computer programming in summer coding academies designed to develop technology leaders.
- [Hidden Genius Project](#) trains African–American young men in technology creation, entrepreneurship, and leadership skills in order to transform their communities and create career pathways.
- [Latinos in Tech Innovation and Social Media](#) aims to empower Latinos in the areas of health, education, and civic engagement through tech innovation.
- [Yes We Code](#) helps prepare youth to become computer programmers, with the goal of training 100,000 young people.

³⁰ While it is beyond the scope of this report to present each program in California, several examples are highlighted as models for increasing computer science access and opportunity.

University-Level Initiatives

- [Computer science researchers](#) from Santa Clara University received National Science Foundation funding to establish Exploring Computer Science at 10 public high schools in the San Jose area. Results show an encouraging correlation for Exploring Computer Science participants versus their peers in attendance rates and mathematics test scores.
- University of California, Berkeley instructors who developed the popular “Beauty and Joy of Computing” (BJC) class have created an [online version](#) of the course, called BJCx, launching September 2015. BJC was twice chosen as a national

pilot for the upcoming Advanced [Placement Computer Science Principles](#) course designed to broaden participation in groups traditionally underrepresented in computing.

- [Computing Principles for All Students’ Success](#), a collaboration between The University of California, San Diego, San Diego State University, the San Diego chapter of the Computer Science Teachers Association, and K-12 schools throughout San Diego, aims to build local capacity and sustained professional development for a regional community of high school Computer Science Principles teachers.

Policy and Advocacy

- [Expanding Computing Education Pathways Alliance](#), of which California is a state partner, seeks to increase the number and diversity of students in the educational pipeline to computing by supporting state-level computing education reforms.
- In September 2014, [California Governor Jerry Brown signed bills AB 1764, SB 1200 and AB 1539 in support of expanding computer science education](#). AB 1764 would allow California schools districts to award students math credit for a UC-approved course in computer science. SB 1200 calls on the University of California and California State University systems to develop guidelines for high school computer science courses that would satisfy advanced math subject matter requirement for undergraduate admissions. AB 1539 calls on the Instructional Quality Commission to consider developing K-12 computer science content standards. New legislation introduced in 2015 includes district grant funding programs for computer science coursework and professional development, a community college concurrent computer science enrollment initiative, and a proposed “Women and Girls in STEM” Week to encourage and celebrate women in the fields of science, technology, engineering, and mathematics.
- [Alliance for California Computing Education for Students and Schools](#) (ACCESS) is a statewide network of computer scientists, K-12 teachers, professors, educational policy advocates, and industry professionals dedicated to providing all California students with high-quality computer science education, ensuring that computer science

education is available to all students, specifically for traditionally underrepresented students including girls, low-income students, and students of color. ACCESS is engaged in tracking, supporting, and monitoring the implementation of bills and ensuring that California’s computer science education legislation will fulfill its potential for expanding participation in computer science and ensuring its accessibility to all students.



Photo © Liz Acosta 2012.

```
6 import flash.system.Security; 7 import flash.net.*; 8 import org.iotashan.oauth.*; 9 10 import net.systemedD.halcyon.MapEvent; 11 import net.systemedD.halcyon.ExtendedURLLoader; 12 import net.systemedD.halcyon.connection.bboxes.*; 13 14 15 /** 16 * XMLConnection provides all the methods required to a live 17 * OSM server. See OSMConnection for connecting to a read-only .osm file 18 * 19 * @see OSMConnection 20 */ 21 public class XMLConnection extends XMLBaseConnection { 22 23     private const MARGIN:Number=0.05; 24     private static var defaultTags:Array=["created_by", 25     "tiger:upload_uid", "tiger:tlid", "tiger:source", "tiger:separated", "tiger:datasetName", "geobase:uid", "sub_sea:type", 26     "odbl", "odbl:note", 27     "yh:LINE_NAME", "yh:LINE_NUM", "yh:TOTYUMONO", 28     "yh:TYPE", "yh:WIDTH_RANK", "SK53_bulk:load"]; 29 30 31     /** 32 * Create a new XML connection 33 * 34 * @param name The name of the connection 35 * @param api The url of the OSM API server, e.g. http://api06.dev.openstreetmap.org/api/0.6/ 36 * @param policy The url of the flash crossdomain policy to load, e.g. http://api06.dev.openstreetmap.org/api/crossdomain.xml 37 * @param initparams Any further parameters for the connection, such as the API key 38 */ 39     public function XMLConnection(name:String,api:String,policy:String,initparams:Object) { 40 41         super(name,api,policy,initparams); 42         if (policyURL != "") Security.loadPolicyFile(policyURL); 43 44         var oauthPolicy:String = getParam("oauth_policy", ""); 45         if (oauthPolicy != "") Security.loadPolicyFile(oauthPolicy); 46     } 47 48     /** 49 * Load a bounding box 50 * 51 * @param left The left edge of the bounding box 52 * @param right The right edge of the bounding box 53 * @param top The top edge of the bounding box 54 * @param bottom The bottom edge of the bounding box 55 * 56 * @return An array of boxes 57 */ 58     public function loadBbox(left:Number,right:Number,top:Number,bottom:Number):void { 59         purgeIfExists(left,right,top,bottom); 60         var requestBox:Box=new Box().fromBbox(left,bottom,right,top); 61         var boxes:Array=fetchSet.getBoxes(requestBox,MAX_BBOXES); 62     } 63 64     /** 65 * Enlarge a bounding box by a given margin on each edge 66 * 67 * @param box The bounding box to enlarge 68 * @param margin The margin to add to each edge 69 * 70 * @return The enlarged bounding box 71 */ 72     private function enlargeBbox(box:Box,margin:Number):Box { 73         var xm:Number=box.right-box.left; 74         var ym:Number=box.top-box.bottom; 75         var left:Number=box.left-xmargin; 76         var right:Number=box.right+ym; 77         var top:Number=box.top+ym; 78         var bottom:Number=box.bottom-ym; 79         return new Box().fromBbox(left,bottom,right,top); 80     } 81 82     /** 83 * Send a request to the OSM API 84 * 85 * @param request The request to send 86 * 87 * @return The response 88 */ 89     private function sendRequest(request:URLRequest):void { 90         var loader:URLLoader=new URLLoader(); 91         loader.addEventListener(Event.COMPLETE,loadedMap); 92         loader.addEventListener(Event.IO_ERROR,errorHandler); 93         loader.addEventListener(HTTPStatusEvent.HTTP_STATUS,mapLoadStatus); 94         loader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,errorHandler); 95         loader.requestHeaders.push(new URLRequestHeader("X-Error-Format","XML")); 96         loader.load(request); 97     } 98 99     /** 100 * Error handler for the OSM API 101 * 102 * @param event The event 103 * 104 * @return void 105 */ 106     private function errorHandler(event:Event):void { 107         dispatchEvent(new Event(LoadEvent.LOAD_ERROR)); 108         dispatchEvent(new Event(LoadEvent.LOAD_ERROR_MESSAGE)); 109         dispatchEvent(new Event(LoadEvent.LOAD_ERROR_MESSAGE)); 110     } 111 112     /** 113 * Map load status handler 114 * 115 * @param event The event 116 * 117 * @return void 118 */ 119     private function mapLoadStatus(event:HTTPStatusEvent):void { 120         if (event.status == HTTPStatus.HTTP_200) { 121             dispatchEvent(new Event(LoadEvent.LOAD_COMPLETED)); 122         } else { 123             dispatchEvent(new Event(LoadEvent.LOAD_ERROR)); 124             dispatchEvent(new Event(LoadEvent.LOAD_ERROR_MESSAGE)); 125             dispatchEvent(new Event(LoadEvent.LOAD_ERROR_MESSAGE)); 126         } 127     } 128 129     /** 130 * Set the access token 131 * 132 * @param id The access token 133 * 134 * @return void 135 */ 136     private function setAuthToken(id:Object):void { 137         authToken = OAuthConsumer.getAuthToken(id); 138     } 139 140     /** 141 * Get the access token 142 * 143 * @return The access token 144 */ 145     private function getAuthToken():OAuthConsumer { 146         if (authToken == null) { 147             var key:String = getParam("oauth_token", null); 148             var secret:String = getParam("oauth_token_secret", null); 149             if (key != null && secret != null) { 150                 authToken = OAuthConsumer.getAuthToken(key, secret); 151             } 152         } 153         return authToken; 154     } 155 156     /** 157 * Get the consumer 158 * 159 * @return The consumer 160 */ 161     private function getConsumer():OAuthConsumer { 162         if (appID == null) { 163             var key:String = getParam("oauth_consumer_key", null); 164             var secret:String = getParam("oauth_consumer_secret", null); 165             if (key != null && secret != null) { 166                 appID = OAuthConsumer.getAppID(key, secret); 167             } 168         } 169         return appID; 170     } 171 172     /** 173 * Create a changeset 174 * 175 * @param tags The tags for the changeset 176 * 177 * @return The changeset 178 */ 179     private function createChangeset(tags:Object):void { 180         lastUploadedChangeset = null; 181         var changesetXML:XML = <osm version="0.6"><changeset /></osm>; 182         var changeset:XML = <changeset />; 183         for each (var tagKey:Object in tags) { 184             var tagXML:XML = <tag />; 185             tagXML.@k = tagKey; 186             tagXML.@v = tags[tagKey]; 187             changesetXML.appendChild(tagXML); 188         } 189         sendOAuthPut(apiBaseURL+"changeset", changesetXML, changesetCreateComplete, changesetCreateError, recordStatus); 190     } 191 192     /** 193 * Changeset create complete handler 194 * 195 * @param event The event 196 * 197 * @return void 198 */ 199     private function changesetCreateComplete(event:Event):void { 200         var result:String = URLLoader(event.target).data; 201         var id:Number = Number(URLLoader(event.target).data); 202         // response should be a Number changeset id 203         // which means we now have a new changeset! 204         setActiveChangeset(new Changeset(this, id, addedChangesetTags)); 205     } 206 207     /** 208 * Changeset create error handler 209 * 210 * @param event The event 211 * 212 * @return void 213 */ 214     private function changesetCreateError(event:IOErrorEvent):void { 215         dispatchEvent(new Event(LoadEvent.LOAD_ERROR)); 216     } 217 218     /** 219 * Close a changeset 220 * 221 * @param changeset The changeset to close 222 * 223 * @return void 224 */ 225     private function closeChangeset(changeset:Changeset):void { 226         var cs:Changeset = getActiveChangeset(); 227         if (cs == changeset) { 228             setActiveChangeset(null); 229         } 230     } 231 232     /** 233 * OAuth request handler 234 * 235 * @param method The method 236 * @param url The url 237 * @param sig The signature 238 * @param resultType The result type 239 * 240 * @return The response 241 */ 242     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 243         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 244         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 245         request.setURL(url); 246         request.setXML(xml); 247         request.send(onComplete,onError); 248     } 249 250     /** 251 * OAuth request handler 252 * 253 * @param method The method 254 * @param url The url 255 * @param sig The signature 256 * @param resultType The result type 257 * 258 * @return The response 259 */ 260     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 261         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 262         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 263         request.setURL(url); 264         request.setXML(xml); 265         request.send(onComplete,onError); 266     } 267 268     /** 269 * OAuth request handler 270 * 271 * @param method The method 272 * @param url The url 273 * @param sig The signature 274 * @param resultType The result type 275 * 276 * @return The response 277 */ 278     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 279         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 280         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 281         request.setURL(url); 282         request.setXML(xml); 283         request.send(onComplete,onError); 284     } 285 286     /** 287 * OAuth request handler 288 * 289 * @param method The method 290 * @param url The url 291 * @param sig The signature 292 * @param resultType The result type 293 * 294 * @return The response 295 */ 296     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 297         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 298         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 299         request.setURL(url); 300         request.setXML(xml); 301         request.send(onComplete,onError); 302     } 303 304     /** 305 * OAuth request handler 306 * 307 * @param method The method 308 * @param url The url 309 * @param sig The signature 310 * @param resultType The result type 311 * 312 * @return The response 313 */ 314     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 315         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 316         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 317         request.setURL(url); 318         request.setXML(xml); 319         request.send(onComplete,onError); 320     } 321 322     /** 323 * OAuth request handler 324 * 325 * @param method The method 326 * @param url The url 327 * @param sig The signature 328 * @param resultType The result type 329 * 330 * @return The response 331 */ 332     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 333         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 334         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 335         request.setURL(url); 336         request.setXML(xml); 337         request.send(onComplete,onError); 338     } 339 340     /** 341 * OAuth request handler 342 * 343 * @param method The method 344 * @param url The url 345 * @param sig The signature 346 * @param resultType The result type 347 * 348 * @return The response 349 */ 350     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 351         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 352         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 353         request.setURL(url); 354         request.setXML(xml); 355         request.send(onComplete,onError); 356     } 357 358     /** 359 * OAuth request handler 360 * 361 * @param method The method 362 * @param url The url 363 * @param sig The signature 364 * @param resultType The result type 365 * 366 * @return The response 367 */ 368     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 369         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 370         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 371         request.setURL(url); 372         request.setXML(xml); 373         request.send(onComplete,onError); 374     } 375 376     /** 377 * OAuth request handler 378 * 379 * @param method The method 380 * @param url The url 381 * @param sig The signature 382 * @param resultType The result type 383 * 384 * @return The response 385 */ 386     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 387         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 388         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 389         request.setURL(url); 390         request.setXML(xml); 391         request.send(onComplete,onError); 392     } 393 394     /** 395 * OAuth request handler 396 * 397 * @param method The method 398 * @param url The url 399 * @param sig The signature 400 * @param resultType The result type 401 * 402 * @return The response 403 */ 404     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 405         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 406         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 407         request.setURL(url); 408         request.setXML(xml); 409         request.send(onComplete,onError); 410     } 411 412     /** 413 * OAuth request handler 414 * 415 * @param method The method 416 * @param url The url 417 * @param sig The signature 418 * @param resultType The result type 419 * 420 * @return The response 421 */ 422     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 423         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 424         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 425         request.setURL(url); 426         request.setXML(xml); 427         request.send(onComplete,onError); 428     } 429 430     /** 431 * OAuth request handler 432 * 433 * @param method The method 434 * @param url The url 435 * @param sig The signature 436 * @param resultType The result type 437 * 438 * @return The response 439 */ 440     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 441         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 442         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 443         request.setURL(url); 444         request.setXML(xml); 445         request.send(onComplete,onError); 446     } 447 448     /** 449 * OAuth request handler 450 * 451 * @param method The method 452 * @param url The url 453 * @param sig The signature 454 * @param resultType The result type 455 * 456 * @return The response 457 */ 458     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 459         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 460         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 461         request.setURL(url); 462         request.setXML(xml); 463         request.send(onComplete,onError); 464     } 465 466     /** 467 * OAuth request handler 468 * 469 * @param method The method 470 * @param url The url 471 * @param sig The signature 472 * @param resultType The result type 473 * 474 * @return The response 475 */ 476     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 477         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 478         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 479         request.setURL(url); 480         request.setXML(xml); 481         request.send(onComplete,onError); 482     } 483 484     /** 485 * OAuth request handler 486 * 487 * @param method The method 488 * @param url The url 489 * @param sig The signature 490 * @param resultType The result type 491 * 492 * @return The response 493 */ 494     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 495         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 496         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 497         request.setURL(url); 498         request.setXML(xml); 499         request.send(onComplete,onError); 500     } 501 502     /** 503 * OAuth request handler 504 * 505 * @param method The method 506 * @param url The url 507 * @param sig The signature 508 * @param resultType The result type 509 * 510 * @return The response 511 */ 512     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 513         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 514         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 515         request.setURL(url); 516         request.setXML(xml); 517         request.send(onComplete,onError); 518     } 519 520     /** 521 * OAuth request handler 522 * 523 * @param method The method 524 * @param url The url 525 * @param sig The signature 526 * @param resultType The result type 527 * 528 * @return The response 529 */ 529     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 530         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 531         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 532         request.setURL(url); 533         request.setXML(xml); 534         request.send(onComplete,onError); 535     } 536 537     /** 538 * OAuth request handler 539 * 540 * @param method The method 541 * @param url The url 542 * @param sig The signature 543 * @param resultType The result type 544 * 545 * @return The response 546 */ 546     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 547         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 548         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 549         request.setURL(url); 550         request.setXML(xml); 551         request.send(onComplete,onError); 552     } 553 554     /** 555 * OAuth request handler 556 * 557 * @param method The method 558 * @param url The url 559 * @param sig The signature 560 * @param resultType The result type 561 * 562 * @return The response 563 */ 562     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 563         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 564         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 565         request.setURL(url); 566         request.setXML(xml); 567         request.send(onComplete,onError); 568     } 569 570     /** 571 * OAuth request handler 572 * 573 * @param method The method 574 * @param url The url 575 * @param sig The signature 576 * @param resultType The result type 577 * 578 * @return The response 579 */ 578     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 579         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 580         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 581         request.setURL(url); 582         request.setXML(xml); 583         request.send(onComplete,onError); 584     } 585 586     /** 587 * OAuth request handler 588 * 589 * @param method The method 590 * @param url The url 591 * @param sig The signature 592 * @param resultType The result type 593 * 594 * @return The response 595 */ 594     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 595         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 596         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 597         request.setURL(url); 598         request.setXML(xml); 599         request.send(onComplete,onError); 600     } 601 602     /** 603 * OAuth request handler 604 * 605 * @param method The method 606 * @param url The url 607 * @param sig The signature 608 * @param resultType The result type 609 * 610 * @return The response 611 */ 603     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 611         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 612         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 613         request.setURL(url); 614         request.setXML(xml); 615         request.send(onComplete,onError); 616     } 617 618     /** 619 * OAuth request handler 620 * 621 * @param method The method 622 * @param url The url 623 * @param sig The signature 624 * @param resultType The result type 625 * 626 * @return The response 627 */ 616     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 627         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 628         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 629         request.setURL(url); 630         request.setXML(xml); 631         request.send(onComplete,onError); 632     } 633 634     /** 635 * OAuth request handler 636 * 637 * @param method The method 638 * @param url The url 639 * @param sig The signature 640 * @param resultType The result type 641 * 642 * @return The response 643 */ 632     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 643         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 644         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 645         request.setURL(url); 646         request.setXML(xml); 647         request.send(onComplete,onError); 648     } 649 650     /** 651 * OAuth request handler 652 * 653 * @param method The method 654 * @param url The url 655 * @param sig The signature 656 * @param resultType The result type 657 * 658 * @return The response 659 */ 648     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 659         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 660         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 661         request.setURL(url); 662         request.setXML(xml); 663         request.send(onComplete,onError); 664     } 665 666     /** 667 * OAuth request handler 668 * 669 * @param method The method 670 * @param url The url 671 * @param sig The signature 672 * @param resultType The result type 673 * 674 * @return The response 675 */ 664     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 675         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 676         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 677         request.setURL(url); 678         request.setXML(xml); 679         request.send(onComplete,onError); 680     } 681 682     /** 683 * OAuth request handler 684 * 685 * @param method The method 686 * @param url The url 687 * @param sig The signature 688 * @param resultType The result type 689 * 690 * @return The response 691 */ 680     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 691         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 692         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 693         request.setURL(url); 694         request.setXML(xml); 695         request.send(onComplete,onError); 696     } 697 698     /** 699 * OAuth request handler 700 * 701 * @param method The method 702 * @param url The url 703 * @param sig The signature 704 * @param resultType The result type 705 * 706 * @return The response 707 */ 696     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 707         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 708         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 709         request.setURL(url); 710         request.setXML(xml); 711         request.send(onComplete,onError); 712     } 713 714     /** 715 * OAuth request handler 716 * 717 * @param method The method 718 * @param url The url 719 * @param sig The signature 720 * @param resultType The result type 721 * 722 * @return The response 723 */ 712     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 723         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 724         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 725         request.setURL(url); 726         request.setXML(xml); 727         request.send(onComplete,onError); 728     } 729 730     /** 731 * OAuth request handler 732 * 733 * @param method The method 734 * @param url The url 735 * @param sig The signature 736 * @param resultType The result type 737 * 738 * @return The response 739 */ 728     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 739         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 740         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 741         request.setURL(url); 742         request.setXML(xml); 743         request.send(onComplete,onError); 744     } 745 746     /** 747 * OAuth request handler 748 * 749 * @param method The method 750 * @param url The url 751 * @param sig The signature 752 * @param resultType The result type 753 * 754 * @return The response 755 */ 744     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 755         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 756         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 757         request.setURL(url); 758         request.setXML(xml); 759         request.send(onComplete,onError); 760     } 761 762     /** 763 * OAuth request handler 764 * 765 * @param method The method 766 * @param url The url 767 * @param sig The signature 768 * @param resultType The result type 769 * 770 * @return The response 771 */ 760     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 771         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 772         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 773         request.setURL(url); 774         request.setXML(xml); 775         request.send(onComplete,onError); 776     } 777 778     /** 779 * OAuth request handler 780 * 781 * @param method The method 782 * @param url The url 783 * @param sig The signature 784 * @param resultType The result type 785 * 786 * @return The response 787 */ 776     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 787         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 788         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 789         request.setURL(url); 790         request.setXML(xml); 791         request.send(onComplete,onError); 792     } 793 794     /** 795 * OAuth request handler 796 * 797 * @param method The method 798 * @param url The url 799 * @param sig The signature 800 * @param resultType The result type 801 * 802 * @return The response 803 */ 792     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 803         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 804         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 805         request.setURL(url); 806         request.setXML(xml); 807         request.send(onComplete,onError); 808     } 809 810     /** 811 * OAuth request handler 812 * 813 * @param method The method 814 * @param url The url 815 * @param sig The signature 816 * @param resultType The result type 817 * 818 * @return The response 819 */ 808     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 819         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 820         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 821         request.setURL(url); 822         request.setXML(xml); 823         request.send(onComplete,onError); 824     } 825 826     /** 827 * OAuth request handler 828 * 829 * @param method The method 830 * @param url The url 831 * @param sig The signature 832 * @param resultType The result type 833 * 834 * @return The response 835 */ 824     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 835         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 836         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 837         request.setURL(url); 838         request.setXML(xml); 839         request.send(onComplete,onError); 840     } 841 842     /** 843 * OAuth request handler 844 * 845 * @param method The method 846 * @param url The url 847 * @param sig The signature 848 * @param resultType The result type 849 * 850 * @return The response 851 */ 840     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 851         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 852         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 853         request.setURL(url); 854         request.setXML(xml); 855         request.send(onComplete,onError); 856     } 857 858     /** 859 * OAuth request handler 860 * 861 * @param method The method 862 * @param url The url 863 * @param sig The signature 864 * @param resultType The result type 865 * 866 * @return The response 867 */ 856     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 867         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 868         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 869         request.setURL(url); 870         request.setXML(xml); 871         request.send(onComplete,onError); 872     } 873 874     /** 875 * OAuth request handler 876 * 877 * @param method The method 878 * @param url The url 879 * @param sig The signature 880 * @param resultType The result type 881 * 882 * @return The response 883 */ 864     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 883         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 884         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 885         request.setURL(url); 886         request.setXML(xml); 887         request.send(onComplete,onError); 888     } 889 890     /** 891 * OAuth request handler 892 * 893 * @param method The method 894 * @param url The url 895 * @param sig The signature 896 * @param resultType The result type 897 * 898 * @return The response 899 */ 888     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 899         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 900         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 901         request.setURL(url); 902         request.setXML(xml); 903         request.send(onComplete,onError); 904     } 905 906     /** 907 * OAuth request handler 908 * 909 * @param method The method 910 * @param url The url 911 * @param sig The signature 912 * @param resultType The result type 913 * 914 * @return The response 915 */ 904     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 915         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 916         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 917         request.setURL(url); 918         request.setXML(xml); 919         request.send(onComplete,onError); 920     } 921 922     /** 923 * OAuth request handler 924 * 925 * @param method The method 926 * @param url The url 927 * @param sig The signature 928 * @param resultType The result type 929 * 930 * @return The response 931 */ 920     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 931         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 932         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 933         request.setURL(url); 934         request.setXML(xml); 935         request.send(onComplete,onError); 936     } 937 938     /** 939 * OAuth request handler 940 * 941 * @param method The method 942 * @param url The url 943 * @param sig The signature 944 * @param resultType The result type 945 * 946 * @return The response 947 */ 936     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 947         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 948         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 949         request.setURL(url); 950         request.setXML(xml); 951         request.send(onComplete,onError); 952     } 953 954     /** 955 * OAuth request handler 956 * 957 * @param method The method 958 * @param url The url 959 * @param sig The signature 960 * @param resultType The result type 961 * 962 * @return The response 963 */ 952     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 963         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 964         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 965         request.setURL(url); 966         request.setXML(xml); 967         request.send(onComplete,onError); 968     } 969 970     /** 971 * OAuth request handler 972 * 973 * @param method The method 974 * @param url The url 975 * @param sig The signature 976 * @param resultType The result type 977 * 978 * @return The response 979 */ 968     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 979         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 980         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 981         request.setURL(url); 982         request.setXML(xml); 983         request.send(onComplete,onError); 984     } 985 986     /** 987 * OAuth request handler 988 * 989 * @param method The method 990 * @param url The url 991 * @param sig The signature 992 * @param resultType The result type 993 * 994 * @return The response 995 */ 984     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 995         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 996         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 997         request.setURL(url); 998         request.setXML(xml); 999         request.send(onComplete,onError); 1000     } 1001 1002     /** 1003 * OAuth request handler 1004 * 1005 * @param method The method 1006 * @param url The url 1007 * @param sig The signature 1008 * @param resultType The result type 1009 * 1010 * @return The response 1011 */ 1000     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 1011         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1012         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1013         request.setURL(url); 1014         request.setXML(xml); 1015         request.send(onComplete,onError); 1016     } 1017 1018     /** 1019 * OAuth request handler 1020 * 1021 * @param method The method 1022 * @param url The url 1023 * @param sig The signature 1024 * @param resultType The result type 1025 * 1026 * @return The response 1027 */ 1016     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 1027         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1028         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1029         request.setURL(url); 1030         request.setXML(xml); 1031         request.send(onComplete,onError); 1032     } 1033 1034     /** 1035 * OAuth request handler 1036 * 1037 * @param method The method 1038 * @param url The url 1039 * @param sig The signature 1040 * @param resultType The result type 1041 * 1042 * @return The response 1043 */ 1032     private function sendOAuthPut(url:String,xml:XML,onComplete:Function,onError:Function):void { 1043         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1044         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1045         request.setURL(url); 1046         request.setXML(xml); 1047         request.send(onComplete,onError); 1048     } 1049 1050     /** 1051 * OAuth request handler 1052 * 1053 * @param method The method 1054 * @param url The url 1055 * @param sig The signature 1056 * @param resultType The result type 1057 * 1058 * @return The response 1059 */ 1048     private function sendOAuthGet(url:String,xml:XML,onComplete:Function,onError:Function):void { 1059         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1060         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1061         request.setURL(url); 1062         request.setXML(xml); 1063         request.send(onComplete,onError); 1064     } 1065 1066     /** 1067 * OAuth request handler 1068 * 1069 * @param method The method 1070 * @param url The url 1071 * @param sig The signature 1072 * @param resultType The result type 1073 * 1074 * @return The response 1075 */ 1064     private function sendOAuthDelete(url:String,xml:XML,onComplete:Function,onError:Function):void { 1075         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1076         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1077         request.setURL(url); 1078         request.setXML(xml); 1079         request.send(onComplete,onError); 1080     } 1081 1082     /** 1083 * OAuth request handler 1084 * 1085 * @param method The method 1086 * @param url The url 1087 * @param sig The signature 1088 * @param resultType The result type 1089 * 1090 * @return The response 1091 */ 1080     private function sendOAuthPost(url:String,xml:XML,onComplete:Function,onError:Function):void { 1091         var request:OAuthRequest=new OAuthRequest(method,url,null,getActiveConsumer(),getAuthToken()); 1092         request.setRequest(sig,OAuthRequest.RESULT_TYPE_URL_STRING); 1093         request.setURL(url); 1094         request.setXML(xml); 1095         request.send(onComplete,onError); 1096     } 1097 1098     /** 1099 * OAuth request handler 1100 * 1101 * @param method The method 1102 * @param url The url 1103 * @param sig The signature 1104 * @param resultType The result type 
```